

# Java程序员 职场全攻略

——从小工到专家

吴亚峰 著

懵懂菜鸟如何才能成长为一名成功的开发人员？

如何在“血雨腥风”的求职大战中脱颖而出？

职场中只要技术过硬就可以吃遍天下吗？

是学招式重要，还是练内功重要？

掌握了哪些技能，才算是真正的高手？

.....

资深技术专家，娓娓道来。





## 本书精彩内容导读：

- ◎初窥门径——行业揭秘
- ◎下山之路——有备无患
- ◎步入江湖——做事的学问
- ◎百尺竿头，更进一步
- ◎天下功夫出少林
- ◎没有必杀技，怎么敢出来混
- ◎武学奥义
- ◎雾里看花——职场误区
- ◎必须通关的游戏——求职之旅
- ◎立足江湖——做人的学问
- ◎江湖多歧路
- ◎几种自废武功的做法
- ◎新锐兵器谱
- ◎杂项

作者是一位有才气的人，也是一位技术牛人。他给大家奉献了一本生动有趣、读来轻松活泼、让人大呼过瘾的作品。他不仅告诉我们如何学习Java，更重要的是告诉我们如何能更好地在程序开发这个行业发展得更好，实现个人价值。

——中软集团SOA架构师 王鑫磊

当作者让我给这本书写点评价时，我还真不确定是否应该这么做。但当我看过内容后，我的一切担心都烟消云散了。这本书内容很好，对于那些打算要在Java开发领域发展的人非常有用。这本书和一般的技术图书不同，它更加侧重于从职场角度来探讨Java程序员需要具备的知识和经验。

——用友软件 ERP开发工程师 李迪锋

怎样从一个Java新手过渡到资深工程师？这是每个职场新人的疑惑。我也是从Hello World开始编程的，对其中的辛酸深有体会。看到这本书感觉很亲切，相信“菜Java”们一定能从这本书中受到很多启发。

——北京中科软科技股份有限公司项目经理 段洁男

有人的地方就有江湖，程序员也不例外。如果仅仅很会编程，而不懂得职场的一些奥妙，那也不会得到大的发展。因为现在的系统不可能由一个人独立完成，而必须要有数十人甚至上百人的配合才行。开发类的图书很多，而能给读者介绍职场经验的却很少，《Java程序员职场全攻略——从小工到专家》就很好，给出了很多有益的建议，值得一读。

——广州好易电子联行服务有限公司技术总监 纪超



责任编辑：胡辛征  
责任美编：李玲

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。



上架建议：Java

ISBN 978-7-121-10246-2



9 787121 102462 >

定价：49.00元



# Java程序员 职场全攻略

——从小工到专家

电子工业出版社  
Publishing House of Electronics Industry  
北京•BEIJING



## 内 容 简 介

本书以包罗万象的 IT 大江湖为背景,将 Java 职场中从入门前的学校菜鸟成长为技术大牛的过程展现给读者,内容饱满但又不失趣味性。在本书中既有入职前的行业探秘、误区排除,也有入职后的口诀传授和江湖新锐兵器介绍。不仅与读者朋友们探讨了 Java 江湖中做事的学问,还探讨了一些职场中做人的道理,可以说是本书是 Java 开发人员的职场宝典。

本书适合于尚在学校对前途感到迷茫的大学生,同时也是初入职场的菜鸟不可多得的修炼指南。对于那些已经在 IT 江湖闯荡数年的老鸟,本书也提供了很多新的思路与策略。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有,侵权必究。

## 图书在版编目(CIP)数据

Java 程序员职场全攻略:从小工到专家 / 吴亚峰著. 北京:电子工业出版社, 2010.2  
ISBN 978-7-121-10246-2

I. J… II. 吴… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字(2010)第 010236 号

责任编辑:胡辛征

文字编辑:江 立

印 刷:北京智力达印刷有限公司

装 订:三河市皇庄路通装订厂

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1092 1/16 印张:25 字数:720 千字

印 次:2010 年 2 月第 1 次印刷

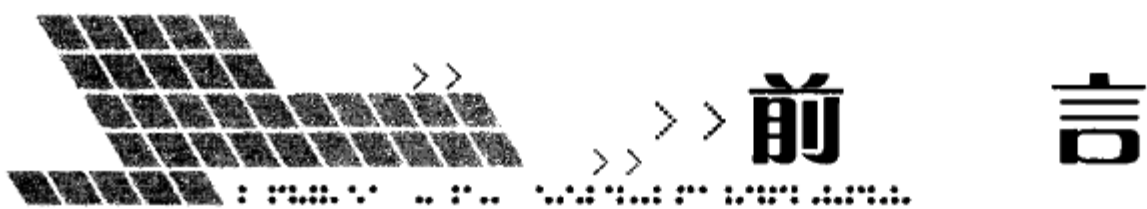
印 数:4000 册 定价:49.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn,盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。





每当坐在自己的工作室里看着书架上林林总总的书籍，使用着高性能的工作站进行开发时，我都会回忆起 1998 年自己刚刚从事 Java 编程时的探索和艰辛。回想当初，仅仅有一台主频 133MHz 的组装机，就敢在上面进行企业级程序的开发，就敢在上面跑 Oracle。由于缺乏有效的技术指导，同时难以像现在这样获得丰富的技术资料，那时候的学习和开发都是很痛苦的。有时连续熬夜很多天也无法将一个小小的程序调通，开发过程中常常伴随着沮丧和郁闷。

而现在的情况已经大不一样了，目前市面上的 IT 技术书籍可谓是铺天盖地，广大读者已经不再需要担心无书可读，而是该担心如何选本好书了。但是纵观浩瀚书海，几乎无不例外地将技术讲解作为重点，这样就忽视了一些更值得关注的问题。

- 懵懂菜鸟如何才能成长为一名成功的开发人员？
- 如何在“血雨腥风”的求职大战中脱颖而出？
- 职场中只要技术过硬就可以吃遍天下吗？
- 是学招式重要，还是练内功重要？
- 掌握了哪些技能，才算是真正的高手？

目前市面上能够回答以上这些问题的书籍可谓寥寥无几。不得不承认，对于许多未入门的 Java 菜鸟和已在 Java 江湖闯荡数年的开发人员来说，目前最缺少的不是技术书籍，而是职业生涯的规划指南及项目开发思想与经验的分享。

多年的 Java 开发经历对我自己来说是一笔宝贵的财富，现在有幸将这些闪光的宝贝写进书中。希望那些曾经启发和指导过自己的不论是技术还是非技术上的点点滴滴，也能够和广大读者朋友产生一些共鸣，更好地帮助大家在 Java 的海洋里乘风破浪。或许海子的那句诗最能表达本书的写作意图：“那幸福的闪电告诉我的，我将告诉每一个人。”

## 本书内容及特色

本书中有两位主人公：职场新手蔡佳娃和业内高人牛开复。而本书也主要以蔡佳娃的成长历程为主线，通过向读者讲述笨手笨脚的蔡佳娃如何在师兄牛开复的帮助下一步一步成长为独当一面的开发人员，把从菜鸟到大牛这个过程像电影般立体地展示给读者，这其实就是我自己的成长经历。

本书力求将 Java 开发人员从菜鸟到大牛成长过程中的方方面面都呈现出来，在内容的组织上花费了较长时间。本书分为上下两篇，上篇主要讲述了如何从 Java 职场的门外汉成功杀入这个英雄辈出的江湖并在其中安身立命、功成名就。其中既有对职场现状的揭秘及行业误区的排除，也有着求职游戏中的详细攻略指南，同时介绍了在职场这个复杂环境中做人和做事的学问。这虽



然都是些技术之外的东西，但是却在很大程度上决定了一名开发人员在自己的“职场”上能走多远。

本书的下篇则主要将自己从事 Java 开发十余年来积累的心得体会分门别类地进行了介绍。有一些是在开发工作中必须精通和掌握的知识，有一些则是从多年编程实践中总结出的“必杀技”，还有一些是在工作中必须杜绝的错误做法。但愿这些技术积累能够帮助读者朋友们在工作中实现鲤鱼跃龙门式的提升，同时也希望那些曾让自己迷茫和困惑的地方，不再成为广大读者朋友前进道路上的绊脚石。

IT 行业是个瞬息万变、发展极快的行业，各种新技术层出不穷，跟不上技术车轮的转动就会在前进的道路上与他人渐行渐远，最终彻底消失在大家的视线中。本书中有一章专门介绍当前 Java 技术中的最新行情，对其或复杂或高深的原理做了简约而完整的介绍，并相应地附以经典示例。读者朋友们可以了解当前江湖中的新锐兵器，并据此确定自己的努力方向。

在编写的过程中，为了让不同水平的读者都能看懂，本书尽量做到讲解通俗，把一些高高在上的技术思想及原理翻译成菜鸟也能看懂的语言，这的确不是一项轻松的工作。在语言的斟酌上，也花费了较多的精力，这是为了让读者朋友不会在阅读的时候感到 boring。读者若能从本书生动却不乏深刻的语言中感受到学习的乐趣和享受之处，便是对本书的莫大鼓励了。

这本书虽然主要探讨的是从事 Java 开发的心得体会，但是一些程序的示例还是要有的。为了不让本书落入一般技术书籍的俗套，在举例的时候尽量保证其简约和经典。因为本书最主要的目的不是让读者朋友们学会表面的技术，而是让大家领悟其中的思想。学习招式在次，提升内力才是最主要的。

经过几个月见缝插针式的奋战，本书终于要交稿了。回顾写书的这几个月时间，不禁为自己能最终完成这个耗时费力的“大制作”而感到欣慰。同时也为自己能将从事 Java 开发多年积累的宝贵经验以及编程感悟分享给正在开发阵线上埋头苦干的广大 Java 人而感到高兴。

贾岛的《剑客》一诗有言：“十年磨一剑，霜刃未曾试，今日把示君，谁有不平事？”，从 1998 年首次接触 Java 算起，到现在也是十年有余。笔者希望用十年的知识和经验磨出的利剑能够帮助广大读者朋友在实际工作中披荆斩棘、奋勇杀敌。

## 本书面向的读者

- Java 初学者

对于未入门的菜鸟，本书首先介绍了当今 IT 职场中 Java 的地位以及 Java 开发人员的生存现状，并为在校大学生指出了一条从学校到职场的明路。这些内容非常有助于读者朋友快速地了解 Java 职场并准确定位自己，同时本书介绍的一些求职攻略也可以为求职者成功挺入职场保驾护航。

- 有一定基础的 Java 开发人员

对于已经在职场立足的新手，本书逐一列举了在职场中 Java 开发人员所必知必会的能力和和一些鲜为人知的编程心法与口诀。除了这些高手之路，本书还指出了一些在工作中会让自己武功全废的错误做法。同时，本书还和读者探讨了在职场中如何更好地为人处世的问题。

- 高级开发人员

对于职场中的高级开发人员，本书所介绍的一些进行 Java 高级开发的技巧和有关新技术的讨

论非常适合高手之间进行分享和交流。高级开发人员也可以通过本书拾遗补漏，在技术水平上精益求精。

## 关于作者

吴亚峰，毕业于北京邮电大学，后留学澳大利亚卧龙岗大学取得硕士学位。1998年开始从事Java开发，有十多年的Java开发与培训经验，主要研究方向是Java EE。现在为Java EE独立软件开发工程师，同时兼任Sun授权Java培训中心认证教师，为数十家著名企业培训了上千名高级软件开发人员，曾编写过《精通NetBeans——Java桌面、Web与企业级程序开发详解》、《Java SE 6.0编程指南》及《30天学通Java项目案例开发》、《30天学通Java Web项目案例开发》等畅销技术书籍。

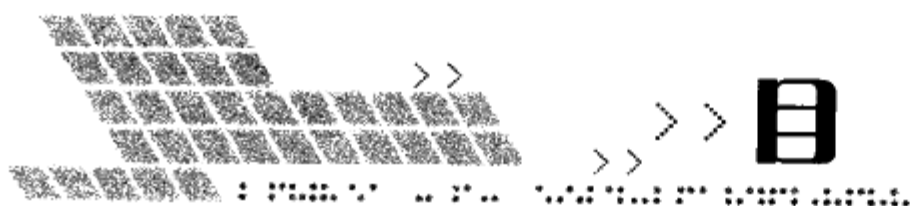
另外，昊燃、方振宇、陈冠佐、傅奎、陈勤、梁洋洋、毕梦飞、陈庆、柴相花、陈非凡、陈华、陈嵩、承卓、陈先在也参与了本书的编写，在此表示感谢！

## 致谢

本书在编写过程中得到了唐山百纳科技有限公司Java培训中心的大力支持，同时魏鹏飞、王海峰、苏亚光以及作者的家人为本书的编写提供了很多帮助，在此表示衷心感谢！

本书作者  
2009年11月





# 录

## 上篇 我与江湖

### 第1章 初窥门径——行业揭秘

2

1.1 IT精英在中国的生存现状.....2	
1.1.1 外行人眼中的IT人.....2	
1.1.2 IT行情分布.....5	
1.1.3 IT语言平台.....7	
1.1.4 你说我容易吗.....10	
1.1.5 我挨踢我骄傲.....12	
1.2 当今主流公司的企业文化.....12	
1.2.1 欧美企业的特色文化.....12	
1.2.2 日韩企业的工作模式.....14	
1.2.3 中资企业的传统特色.....15	
1.2.4 两种不同的软件外包方式.....16	
1.2.5 加入什么样的公司.....18	

1.3 散兵游勇还是团队作战.....18	
1.3.1 哪样多一些.....18	
1.3.2 团队和单兵.....19	
1.3.3 不要停止思考.....20	
1.4 这条路大家都是怎么走的.....20	
1.4.1 职位和待遇是怎么升的.....20	
1.4.2 有干不动的时候吗.....23	
1.4.3 走的人多了，还会有路吗.....24	
1.5 大公司，小公司.....26	
1.5.1 大公司爱专才.....26	
1.5.2 小公司爱多面手.....27	
1.6 本章小结.....28	

### 第2章 雾里看花——职场误区

29

2.1 到底差不差钱.....29	
2.1.1 大家都是个什么身价.....29	
2.1.2 给自己估个好价.....32	
2.1.3 先挣的是资本，后挣的是钱.....34	
2.2 谁给我解决户口问题.....35	
2.2.1 讲讲户口的故事.....35	
2.2.2 各地户口政策面面观.....37	
2.2.3 别怕，咱有暂住证呢.....39	
2.2.4 户口问题小结.....40	

2.3 我们不是爱加班.....40	
2.3.1 常态加班是为何.....40	
2.3.2 你为什么加班.....43	
2.3.3 让自己不再加班.....43	
2.4 莫学狗熊掰棒子.....44	
2.4.1 做过的这辈子永远都不会忘吗.....44	
2.4.2 为自己维护一个小仓库.....46	
2.4.3 多写开发心得.....48	
2.5 本章小结.....49	

### 第3章 下山之路——有备无患

50

3.1 从学生升级到开发人员.....50	
3.1.1 学校给了你什么.....50	
3.1.2 咱们还缺啥.....54	
3.1.3 经验，还是经验.....56	
3.2 为自己定下目标.....58	

3.2.1 目标的意义.....58	
3.2.2 树立目标的学问.....59	
3.2.3 让自己知道今天该干什么.....61	
3.3 IT认证的问题.....63	

3.3.1 认证那点事.....	63
3.3.2 现在的认证.....	65

3.3.3 该不该考个证.....	69
3.4 本章小结 .....	70

## 第4章 必须通关的游戏——求职之旅

71

4.1 简历靓起来 .....	71
4.1.1 简历不是这样写的.....	71
4.1.2 写出出色的简历.....	74
4.1.3 如果是机器筛选简历.....	77
4.1.4 简历小结 .....	77
4.2 笔试，混可不行 .....	78
4.2.1 初识笔试 .....	78
4.2.2 牛刀初试 .....	80
4.2.3 笔试小结 .....	83
4.3 面试——最难的 BOSS.....	83

4.3.1 面试面什么 .....	83
4.3.2 支招面试.....	85
4.3.3 面试演习 .....	87
4.3.4 面试小结 .....	92
4.4 试用期——这才是最后一关 .....	92
4.4.1 试用期考查什么 .....	92
4.4.2 多做什么，少做什么 .....	93
4.4.3 试用期小结 .....	94
4.5 本章小结 .....	94

## 第5章 步入江湖——做事的学问

95

5.1 身为菜鸟.....	95
5.1.1 打碎牙齿往肚里咽.....	95
5.1.2 菜鸟不应该自卑.....	97
5.1.3 一叶障目，不见泰山.....	99
5.2 锐意进取，菜鸟无敌 .....	101
5.2.1 既是初生牛犊，就别怕虎.....	101
5.2.2 勤于学习，落后就要挨打.....	104
5.2.3 菜鸟应该懂得的几件事.....	106
5.3 知足常乐，健康心态 .....	109
5.3.1 总有你达不到的高度.....	109
5.3.2 职场爬山论.....	110

5.3.2 做最好的自己.....	112
5.4 菜鸟何以菜，大牛何以牛 .....	113
5.4.1 代码量的问题.....	113
5.4.2 敢于往上走一步 .....	115
5.4.3 升天不成，掉下来也是个 半仙.....	116
5.5 酒香也怕巷子深 .....	117
5.5.1 找到你的优势.....	117
5.5.2 学会竞争.....	118
5.5.3 发展才是硬道理.....	120
5.6 本章小结 .....	121

## 第6章 立足江湖——做人的学问

122

6.1 新环境有新态度 .....	122
6.1.1 开发人员和厨师.....	122
6.1.2 做人是为了做事.....	123
6.2 同事——战友和对手.....	124
6.2.1 竞争与合作中的做人智慧.....	124
6.2.2 做一个好同事.....	127
6.3 上级，不是校长或家长 .....	130
6.3.1 是员工，不是学生.....	130
6.3.2 上级讨厌的员工.....	130
6.3.3 怎样与上级处理好关系.....	135

6.4 新人和下属，曾经的你 .....	136
6.4.1 准备工作.....	136
6.4.2 学着做个好领导.....	137
6.4.3 被夹在自己的上级和下级之间 怎么办.....	139
6.5 客户，领导内行的外行上帝 .....	140
6.5.1 如何招待上帝.....	140
6.5.2 不要这样对待上帝.....	142
6.5.3 如何对付不可能完成的 任务.....	144



6.6 学着处理和 MM 的关系 .....	145
6.6.1 这个行业的男女比例 .....	145

6.6.2 如何面对异性员工 .....	146
6.7 本章小节 .....	146

## 第 7 章 百尺竿头，更进一步

147

7.1 技术不是万能的 .....	147
7.1.1 为何 IT 是个服务业 .....	147
7.1.2 业务流程要清楚 .....	147
7.1.3 专业领域的知识要了解 .....	149
7.1.4 软件系统的操作方式 .....	149
7.2 书是人类进步的阶梯 .....	150
7.2.1 还要不要读书学习 .....	150
7.2.2 选本好书不容易 .....	151
7.3 解决问题的方法 .....	152
7.3.1 正招和歪招 .....	153
7.3.2 优先使用正招 .....	154

7.3.3 正招不够，歪招也可以上 .....	156
7.4 软件产品的目标 .....	159
7.4.1 实现功能是底线 .....	159
7.4.2 提升性能带来质的飞跃 .....	160
7.5 多多参加技术大会和沙龙 .....	164
7.5.1 何为技术大会 .....	164
7.5.2 我们为什么去技术大会 .....	165
7.5.3 技术大会 PK .....	165
7.5.4 技术沙龙 .....	169
7.6 本章小结 .....	169

## 第 8 章 江湖多歧路

170

8.1 “学院”派和“企业”派 .....	170
8.1.1 何为“学院”派 .....	170
8.1.2 “企业”派的实干 .....	173
8.1.3 一起来做“企业”派 .....	176
8.2 关于“剑宗”和“气宗”的讨论 .....	177
8.2.1 何为“剑宗” .....	177
8.2.2 何为“气宗” .....	178
8.2.3 奇技淫巧不如提升修为 .....	180

8.3 有自己的平台才是王道 .....	181
8.3.1 关于框架的纯“拿来主义” .....	181
8.3.2 项目的分割 .....	184
8.4 “大而全”还是“精而深” .....	186
8.4.1 “大而全”和“精而深”矛盾吗 .....	186
8.4.2 “大而全”托出“精而深” .....	188
8.5 本章小结 .....	189

## 下篇 笑傲江湖

## 第 9 章 天下功夫出少林

192

9.1 Java EE 开发人员必知必会 .....	192
9.1.1 坚实的基础——核心 Java .....	192
9.1.2 只会 Java 可不行——大牛的百宝囊 .....	195
9.2 Java ME 开发人员必知必会 .....	197
9.2.1 了解不同平台对 Java ME 的支持 .....	198
9.2.2 游戏开发的基础知识 .....	200
9.2.3 网络编程知识 .....	205

9.2.4 3G、Android 对 Java ME 开发人员的挑战和机遇 .....	206
9.3 当下流行 EE 框架揭秘 .....	207
9.3.1 Struts 和 WebWork 那点事 .....	208
9.3.2 Tapestry 框架 .....	211
9.3.3 Spring——不可多得的好框架 .....	212
9.3.4 Hibernate——从关系世界到对象世界 .....	215

9.4 大型项目青睐的技术与平台 .....	217
9.4.1 JSF 框架 .....	217
9.4.2 EJB 3.0 业务层技术 .....	219
9.4.3 JPA 持久层技术 .....	223
9.4.4 常见应用服务器简介 .....	224

9.4.5 Java 企业平台的荣耀之路 .....	225
9.5 如何学好框架 .....	226
9.5.1 全面了解各项功能 .....	226
9.5.2 彻底研究工作机理 .....	227
9.6 本章小结 .....	228

## 第 10 章 几种自废武功的做法

229

10.1 相信谬论 .....	229
10.1.1 说出来别不信——链表和 数组的速度问题 .....	229
10.1.2 Java 真的比 C/C++ 慢吗 .....	232
10.2 迷信工具，缺乏纯代码能力 .....	234
10.2.1 迷信 ORM .....	235
10.2.2 神化 IDE .....	237
10.3 浅尝辄止，孤陋寡闻 .....	240
10.3.1 finally 的忽视 .....	240
10.3.2 PreparedStatement 的误解 .....	243

10.3.3 管理数据库连接不知连接池 .....	246
10.4 忽视内存管理 .....	250
10.4.1 对象的 3 种引用 .....	251
10.4.2 “小肥猪”问题 .....	255
10.5 看了就不要再犯的错误 .....	256
10.5.1 “+”惹的祸 .....	256
10.5.2 魔法数字 .....	258
10.5.3 代码复制师的渺茫前途 .....	259
10.5.4 老寿星变量 .....	260
10.6 本章小结 .....	262

## 第 11 章 没有必杀技，怎么敢出来混

263

11.1 精通 SQL .....	263
11.1.1 掀起 SQL 的盖头来 .....	263
11.1.2 强大的 SQL .....	265
11.1.3 SQL 优化问题 .....	268
11.1.4 当下主流的数据库产品 .....	270
11.2 拿下正则式 .....	272
11.2.1 细说正则式 .....	273
11.2.2 正则式在 Java 中的运用 .....	277
11.2.3 正则式在 JavaScript 中的 运用 .....	281

11.3 不会用 Ant 的开发人员不是好 Developer .....	284
11.3.1 Why Ant .....	284
11.3.2 Ant 初体验 .....	286
11.4 浅谈设计模式 .....	288
11.4.1 设计模式的重要性 .....	288
11.4.2 MVC 设计模式 .....	289
11.4.3 单例模式 .....	292
11.4.4 最终守护者模式 .....	293
11.5 本章小结 .....	295

## 第 12 章 新锐兵器谱

296

12.1 面向服务的体系架构 (SOA) .....	296
12.1.1 对面的 SOA 看过来 .....	296
12.1.2 零距离接触 Web Service 开发 .....	299
12.1.3 博采众长之集大成者—— CXF .....	302
12.1.4 英雄不问岁数——Axis 2 .....	306
12.1.5 走近 ESB——企业服务总线 .....	309

12.2 富客户端应用 (RIA) .....	313
12.2.1 从平淡到酷炫—— RIA 与 AJAX .....	313
12.2.2 酷炫背后的基石——核心 JavaScript .....	316
12.2.3 AJAX 的开发利器——Dojo .....	319
12.2.4 AJAX 的最酷代表作—— GoogleMap .....	323

12.2.5	Web 2.0 时代的异军突起—— Mashup.....	326
12.2.6	RIA 殿堂的技术新贵—— JavaFX .....	328

12.3	搜索引擎技术 .....	337
12.3.1	Lucene 开源项目 .....	337
12.3.2	Nutch 框架 .....	342
12.4	本章小结 .....	345

## 第 13 章 武学奥义

346

13.1	单元测试的利器——JUnit .....	346
13.1.1	JUnit 简介 .....	346
13.1.2	单枪匹马，赤膊上阵—— JUnit 的单独使用 .....	347
13.1.3	岂曰无衣，与子同袍—— JUnit 和 Ant 的联合 .....	350
13.1.4	得道者多助——JUnit 在 Eclipse 和 NetBeans 中的使用 .....	352
13.2	版本管理 .....	354
13.2.1	版本不可一日不控 .....	354
13.2.2	沙场秋点兵之版本控制系统 .....	355
13.2.3	版本控制系统与 IDE 的协作 .....	356
13.3	UML 建模语言 .....	357
13.3.1	UML 就这么回事 .....	358

13.3.2	UML 之实战 IDE .....	359
13.4	大型服务器操作系统 .....	360
13.4.1	UNIX 平台 .....	360
13.4.2	Linux 平台 .....	361
13.4.3	Windows Server 平台 .....	362
13.5	集群与负载均衡 .....	362
13.5.1	集群 .....	363
13.5.2	幂等操作 .....	364
13.5.3	我们的程序运行在哪 .....	365
13.6	虚拟化与云计算 .....	366
13.6.1	举杯邀明月，对影成三人 ——虚拟化 .....	367
13.6.2	云中谁寄锦书来——云计算 .....	368
13.7	本章小结 .....	370

## 第 14 章 杂项

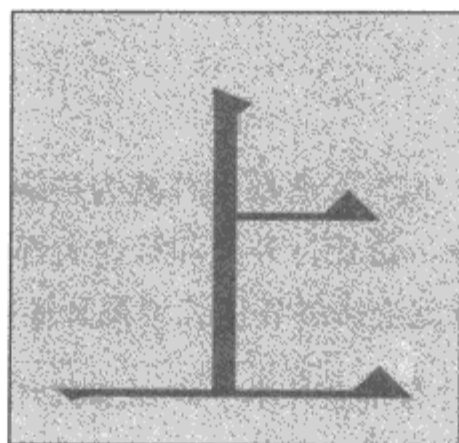
371

14.1	专业英语不能不熟练 .....	371
14.1.1	向高新技术看齐 .....	371
14.1.2	等到中文版的时候 .....	372
14.1.3	做一个大牛的需要 .....	373
14.2	维护大脑这个数据库 .....	373
14.2.1	书到用时方恨少 .....	373
14.2.2	让积累成为一种习惯 .....	374
14.2.3	搜索引擎的使用 .....	375

14.3	IT 人也要不务正业 .....	379
14.3.1	不懂数学岂不是很糟糕 .....	380
14.3.2	谁说物理是白学了 .....	382
14.3.3	一起来不务正业吧 .....	383
14.4	读学术论文 .....	383
14.4.1	别怕我，我是好人 .....	383
14.4.2	醍醐灌顶，如坐春风 .....	385
14.5	本章小结 .....	386



P A R T



---


## 上篇 我与江湖

---

- 第1章 初窥门径——行业揭秘
- 第2章 雾里看花——职场误区
- 第3章 下山之路——有备无患
- 第4章 必须通关的游戏——求职之旅
- 第5章 步入江湖——做事的学问
- 第6章 立足江湖——做人的学问
- 第7章 百尺竿头，更进一步
- 第8章 江湖多歧路

# 第 1 章 初窥门径——行业揭秘

IT, 即 Information Technology, 也有人恶搞地称其为挨踢。这是一个诞生没多少年的新兴产业, 但却是个英雄和天才辈出的行业, 这让行外人和行内人都十分赞叹。这个充满科技天才和财富的世界, 究竟是什么样子呢? 本章就会带领各位读者揭开糊在 IT 行业上的这层纸。

 **显示** IT 含义很广泛, 统指所有信息技术革命以来的计算机和电子通信技术。而本书中所指的 IT, 一般只指计算机软件开发这一领域。

与读者同时开始 IT 职场之旅的, 还有本书的两位主人公:

- 蔡佳娃, 男, ××理工大学计算机专业大三学生, 做事比较踏实, 小心谨慎, 好钻研, 但经常门路不对。
- 牛开复, 男, 毕业于××理工大学计算机专业, 曾就业于 IBM 中国研发中心, 目前在北京从事软件开发, 业内高人。

## 1.1 IT 精英在中国的生存现状

IT 在中国的大发展不过近 30 年, 但却是风起云涌、豪杰四起的 30 年。在中国的 IT 人中也不可否认地诞生了许多天才和富豪, 但是这里比较关心的还是人数最多的大众 IT 人的生存现状, 毕竟是千千万万的他们盖起了中国的 IT 大厦。

### 1.1.1 外行人眼中的 IT 人

不光是外行人, 就连刚入行不久的新手, 看 IT 这一行都是单纯的仰视和羡慕。面对这个新生而又成长极快并且正在迅速包围每个人生活方方面面的行业, 很多人都是好奇与惊奇并存, 羡慕与向往同在。总结一下, 外行人看 IT 这个行业有如下几个特征:

- 高薪
- 年轻
- 高深
- 神奇
- 工作狂

升入大三的蔡佳娃, 茫然间觉得一切都已迫在眉睫, 大学四年已然过去了一半, 是时候看看外面的世界了。蔡佳娃很想了解一下两年后自己将要杀入的 IT 行业, 于是门外汉的他便找到了已经混迹江湖大有所成的师兄牛开复。

#### 1. 高薪

“牛开复师兄, 你给句实话, 咱们这行算不算高薪, 到底挣多少?”

“呵呵，高薪，或许是你们对这个行业的第一印象吧。我记得2008年中国收入最高的十大职业调查中，IT从业者的收入名列第二位，算是高的了。IT这个行业先是被一道金色的光芒笼罩着，而且长期之内，这也是外行人向IT这边看时感受的第一束光。”

“那也就是真的高薪了？”

“也不能全靠调查，调查从本质上说是有些片面的，因为360行，行行出富豪，这种调查也只是一种总体上的、平均性质的。不过就算是个平均值，IT这个行业还是能引起人们无尽的遐想。试问，谁会不自信到怀疑自己混不到平均值的水平呢？”

#### 业内心声：

- 高薪是没错，但是IT是个贫富差距很大的行业，往往刚刚入门的程序员和高级顾问的收入差距会有几十倍。而中国的IT薪水又是呈两头小、中间大的梭子型分布，拿低薪和高薪的人少，大部分人都在中间徘徊。
- 在高薪的耀眼光芒下，谁也不知道有多少人毅然地奔向了IT的怀抱，但是如果粥还是那点粥，和尚越来越多的话，每个人分到的就会变少。
- 真正的IT精英收入依然较高，只是从菜鸟走向高薪和高手的道路因为同行者的增加而变得更加崎岖不平。不过这样的生存方式却很重要，只有这样严格遵循自然法则地发展，才使得IT这个行业在现在和未来大放异彩。

#### 2. 年轻

“师兄，干这行的是不是都是年轻人哪？”

“何以见得啊？”

“我也不知道，就是一想到IT就觉得是年轻人干的事，可能是受一些网络或电视上媒体的影响吧，反正我还没有见过一个头发花白的开发人员呢。”

“那可不一定哦，IT行业那些学术界的宗师和什么什么之父之类的可都是老先生呢。可能是我国的IT行业起步较晚吧，在我看来这一行的从业人员都比较年轻，不过你可以试试做一个中国年纪最大的开发人员。”

#### 业内心声：

- 中国IT专业人员的年龄主要集中在21~35岁，其中26~30岁比例最高，占到四成；其次是21~25岁人群，略少与前者；31~35岁居第三位，不足两成；剩下的不足半成的是其他年龄段的，大都是35岁以上的开发人员。
- IT这个行业是一个充满激情的行业，所有人都是抱着迎难而上和求知若渴的心态在工作。所以，年轻行业和年轻的从业者，谁造就了谁，谁是因谁是果还真不好说清楚。年轻，或许更多指的还是一种心态。

#### 3. 高深

“师兄啊，我以前，包括现在也是这么觉得，IT这个行业实在是高深莫测。要让死气沉沉的机器按照人的意志来做事，IT这工作真不是一般人可以做得来的。”

“呵呵，这倒不假，干IT这一行对于头脑的要求还真不低，因为软件是个没有实体的东西，



所有东西都必须是在脑海里的模型化存在，唯一面对的也只有洋洋洒洒的代码了。”

“就是啊，光面对那些看不到边的代码就够了，还要研究数学啦、算法啦等一般人几乎这辈子都不会深究的东西，光技术术语就一堆一堆的。”

“你说得没错，有时候我们也觉得要学的知识实在是太多了。不过你慢慢会发现，脑子就像是 BT 下载，用得越多就越好使。”

业内心声：

- 这倒不假，IT 的确是个聪明脑袋的聚集地，是个对个人能力要求比较高的行业，没有真本事会立刻被踢出队伍。IT 研究的是毫无趣味的代码和飘渺的算法，但是创造出来的却是极大方便人们的各种软件，这些软件功能强大，却简单易用。
- IT 人承担了机器语言和使用者的思想之间的翻译工作。其他职业要想培养出一个成手也许不会太难，而 IT 界，尤其是在竞争越来越激烈的形势下，想变成一个企业乐于聘用的合格开发人员，所要下的工夫绝对不仅仅是熬夜就可以达到的。

#### 4. 神奇

“师兄，你知道我为什么学计算机这个专业吗？”

“不知道，脑袋被门挤了？”

“哪啊，我就是觉得 IT 这个行业很酷啊。好多新生名词不都是 IT 技术的发展催生出来的嘛。博客、贴吧、论坛……还有那么多的技术流行语。”

“但是很多技术术语都很深奥啊。”

“这种深奥不比薛定谔方程的那种不想懂也不想关心的深奥，这种深奥有种 fashion 的感觉，呵呵。”

“我懂你的意思啦，IT 给人的感觉很时尚是吧。”

“对喽。这就是我选择学计算机的原因！IT 世界就像个魔法世界，不断创造着让人眼前一亮的技术和流行词汇。”

业内心声：

- 的确是这样，IT 正以前所未有的速度改变着人们的生活，用神奇来说一点也不为过。光看中国网民的增长速度就会发现 IT 在互联网上吸引了多少人。
- 或许是只缘身在此山中吧，IT 人并不觉得这个行业有多时尚。其实也只有 IT 人才明白，神奇的背后可是无数个加班的日夜和费心的思考。当然，还有闪光的 idea。

#### 5. 工作狂

“师兄，这个行业中的人是不是都是工作狂啊，干起活来不要命的？”

“是不是看新闻看的啊？”

“不光是新闻，其实也听其他人讲过，好多人因为不是工作狂而选择了离开这个行业呢。”

“你说得也不算全对，IT 有时候是需要工作玩命一些，因为脑力劳动有时候不能断，一断就接不好，有时候封闭式开发一两个月不见天日。有时候在公司加班还不过瘾，还把工作带回家去做。这些都有可能发生，但是我们玩命更多的是源自激情。”

业内心声:

- 大部分 IT 人都会有一些工作狂, 因为脑力劳动不比体力劳动, 脑力劳动相对不容易疲劳, 尤其再加上不少 IT 从业人员都爱较劲, 所以工作热情一上来, 工作拼点命也就比较常见了。
- 但是有时候工作狂也可能是因为主管交给员工的任务没法按时完成, 或是领导交给主管的任务没法按时完成, 所以这种拼命就很无奈了。

上述几个特征, 大概就是外行人对 IT 人的印象吧。总的来说, 这是一个说起来很有面子的行业, 时髦、多金。同时, 这个行业的人工作起来比较拼命。或许正是这些特征, 吸引着一批又一批的年轻人投入其中, 创造着自己的神话。

### 1.1.2 IT 行情分布

为什么要提一下中国的 IT 行情分布呢? 因为很多有志于从事 IT 行业的人, 对整个 IT 行业在中国的行情并不了解, 单纯地认为只要是种子, 在哪里都会发芽, 却往往忽略了自己职业生涯的成长与目标。例如蔡佳娃同学就是这样。

“蔡佳娃, 毕业了去哪里啊?”

“我只是想着如何自信满满地毕业, 还真没想过该去哪。去哪不都一样吗?”

“怎么会一样呢? 就算是一种花, 也有它适宜生长的地方, 什么地方开得艳, 什么地方开得蔫, 什么地方不会开花, 什么地方年年开花还节节高。”

“师兄你是说我是那株花?”

“不光你是, 整个 IT 行业也是啊。”

“我觉得堂堂一个 IT 产业, 不该会有地区的差异和限制吧?”

“不仅有, 而且是相当大啊。所以, 在你还没有实力改变土质的时候, 最好选择一块好的土壤。”

“那师兄你就给我讲讲去哪扎根吧!”

本节就来研究 IT 这朵花的生长习性, 这里所指的行情分布, 主要分为以下两点:

- IT 从业人员分布
- IT 服务性质

#### 1. IT 从业人员分布

“先给你看看中国的 IT 从业人员分布表吧, 看看中国的 IT 精英们都聚集在哪里研究问题呢。”  
(见表 1-1)

“哇, 分布太不平衡了。北京、广东和上海的 IT 从业人员几乎就是全国 IT 总人数的一半啊。”

“所以说嘛, 中国的 IT 人才分布还是很不平衡的, 而且就当前情况来看这种状况会持续很长一段时间啊。”

“人多能说明什么啊?”

“IT 最基本的单位就是人才, 所以人才多的地方机会就多, 技术的层次就会较高。而一些从业人员少的地方, 也在一定程度上反映了那个地方的市场需求, 所以机会不多, 也就很少有机会接触新技术了。也就是说蛋糕如果很小, 那就用不着刀子切了, 一口就可以直接吞下去。”

“那选择地区就应该优先选择人多的地方喽?”

“也不尽然, 不排除某一天其他某个地方会出现另一个‘硅谷’的可能哦。比如 IBM 头脑一

热把中国研究院迁到了×××，那可能就是另一个‘硅谷’呢。不过一般情况下，为了自己更好地发展，还是尽量去那些人才和技术比较集中的地方。”

表 1-1 IT从业人员分布表

省 市	IT 从业人员比例	省 市	IT 从业人员比例
北 京	22.4%	河北省	2.3%
广东省	15.8%	四川省	2.1%
上海市	11.5%	湖北省	2.0%
江苏省	7.3%	江西省	1.9%
浙江省	5.2%	天津市	1.8%
辽宁省	4.4%	河南省	1.6%
山东省	3.5%	黑龙江省	1.5%
福建省	3.5%	重庆市	1.3%
陕西省	2.6%	安徽省	1.1%
其他省市	8.2%		

由表 1-1 可知，中国的 IT 从业人员分布一般集中在东部沿海一带，而东部沿海一带又以北京、上海、广东三个地区所占的比重最大，所以这些地方也都是中国 IT 技术的前沿阵地，拥有最高级的开发人员，拥有全国最先进的技术。当然，也存在着全国最激烈的竞争。

## 2. IT 服务性质

“刚才我们谈了 IT 行业地区分布的差异，现在谈一谈服务性质的差异吧。”

“服务性质？”

“对啊，首先你要明确一个概念，IT 产业属于服务业，就像医疗、教育、金融一样。既然是服务业肯定就有服务对象，这里我们不会分得太细，只是将软件开发产业面向的客户群体做一个简单介绍。”

“啊，那师兄你说吧。”

“中国的 IT 产业，一大部分都是‘自产自销’，服务于国内的各个行业，比如电力、银行、钢铁等，我们姑且称之为传统的软件产业。”

“另外一种呢？”

“另外一种也是近几年发展迅速的运营方式，也就是软件外包产业。软件外包充分利用了劳动力成本和全球化的优势，接受主要来自欧美、日韩等劳动力成本较高的公司的外包业务。”

“那中国比较知名的外包公司和地区大概是哪里啊？”

“提到中国的软件外包，不得不提大连，大连是中国最为出色的对外外包基地之一，由于地理位置等得天独厚的有利条件，使其在主要面向日本的 IT 公司外包业务上获得了很大的成功。最著名的软件外包公司东软就在大连。”

“软件外包的发展会很好吗？”

“我个人比较看好软件外包，因为一方面国家政策很扶持，另一方面我国的软件外包优势很明显，软件外包在未来几年肯定会有更好的发展。”

中国传统的软件开发产业也可以分为以下两种开发形式：

- 客户提出需求，在自己的公司做项目开发，项目开发和测试完成后，再拿到客户所在地进行部署调试。这种开发方式应该比较符合职场新手对软件开发的理解。
- 客户提出需求，自己派团队到客户所在地进行项目开发，并随时部署调试。这种开发方式工期短，效率高，反应速度较快，一般用在政府、银行等大型项目中。这种项目一般安全性要求也非常高，所以很多时候都是封闭式开发。

软件外包，就是接到项目后其中的一部分自己不做，承包给别人去做。软件外包中的发包方就是将项目承包出去的一方，是接包方的客户；接包方就是赚钱的一方，负责将分给自己的项目做好。软件外包产业是个非常有前景的产业，中国正在努力让自己成为一个全球性的软件外包基地，就像印度一样。

中国的软件外包主要分布在北京、广东、上海、辽东等地区，其中对日外包所占比例较大，其次是美国和欧洲。随着全球化步伐的加快和中国软件技术水平的提高，中国软件外包竞争力将会大大增强，中国正在成为全球数一数二的软件外包基地。

软件外包开始越来越独立出来，由国家政策的扶持和业内巨头的牵头，出现了很多像大连这样的外包产业基地，如苏州工业园区等。软件外包在聚集的同时，也在向 IT 中心城市的周边发展，因为软件外包本来就是劳动力成本差异而产生的，因此一些周边城市的成本优势就很明显了。

### 1.1.3 IT 语言平台

软件开发这个行业的武器就是编程语言。IT 发展到今天，产生了种类繁多的编程语言，有的古老却仍然实用，有的新生而激情四射，有的只是昙花一现就黯然消逝，有的则从诞生之日起就不断发展壮大。选择何种编程语言平台，便是摆在很多希望步入 IT 行业人面前的首道难题。

“蔡佳娃，说说看，你都学过什么编程语言啊？”

“不算少哦，我学过 C、C++、选修过 VB、下学期还会有 Java。”

“那你准备选哪种语言入行呢？”

“这倒没想过，不过只要学得多了都有好处吧？”

“那可不一定，你学得多我不反对，不过必须得有一门精通的语言，就像种了十盆花，总得有一盆是拿得出手的吧？”

“嗯，那倒是，那师兄你给我介绍介绍当今的编程语言行情呗。”

“OK，先给你看看最新的编程语言排行榜。”（见图 1-1）

图 1-1 列出了 2009 年 9 月编程语言排行前 12 名的柱状分布图，可以看出 Java 虽然优势不是特别大，但还是顽强地占据开发语言老大哥的位子。其他比较强势的语言是 C、PHP、C++ 和 VB，这几门语言占据了所有编程语言的半壁江山还要多。

#### 1. Java

Java 语言自 1995 年发布以来，由于其面向对象、跨平台和分布式的特性迅速风靡整个 IT 界。加之 Sun、Oracle、IBM 等行业巨头的大力支持，以及众多开发者为其设计的各种框架技术，使得 Java 技术涵盖了当今软件开发的几乎所有方面。其中，Java EE 和 Java ME 分别在企业级和移动开发上牢牢占据着霸主的地位。

另外，由于 Java 主要面向上层应用，运行在 Java 虚拟机上，所以无法对系统底层进行很强的



操作。因此底层系统的开发，如操作系统、51 单片机等，一般是不能采用 Java 技术的。

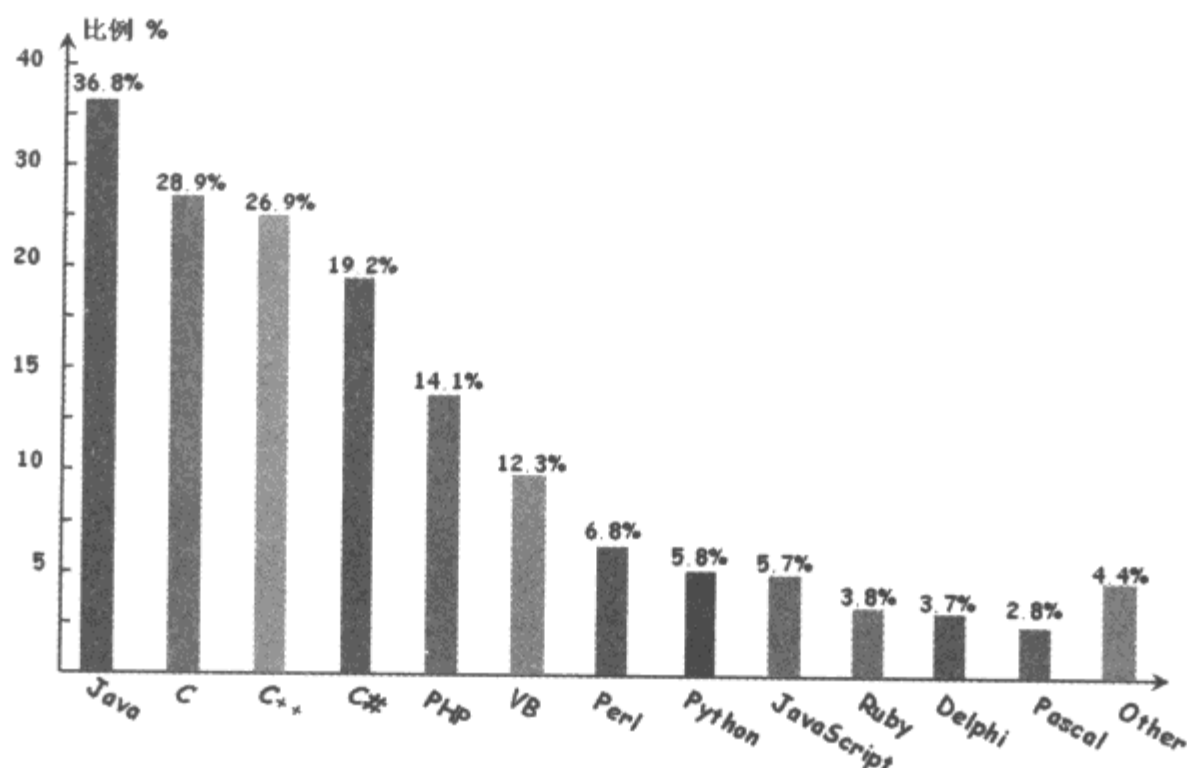


图 1-1 编程语言分布柱状图

## 2. C

C 语言历史悠久，功能强大，很多编程语言都或多或少地受到 C 语言的影响，发展到今天仍然可以傲视大部分后来者。C 语言是一种面向过程的结构化语言，由于其面向底层，编程灵活，效率高，广泛应用在嵌入式、操作系统等系统级别的开发中。

早期 C 语言也用来做上层软件的开发，近些年由于其他专门针对上层开发的语言平台（如 Java）的长足发展，C 语言逐渐淡出了上层软件的开发。

## 3. C++

C++ 衍生自 C 语言，但是发展到现在，C++ 和 C 语言已经独立开来，并且 C++ 编译器也可以容纳 C 的源代码。C++ 很少被用于 Web 级和企业级开发，更多地被应用在系统级开发的项目中，在大型游戏、设备驱动等方面有较大的优势。

但是由于 C++ 语法并没有彻底甩掉 C 语言的累赘，面向对象搞得不够彻底，使得 C++ 语言过于复杂，可靠性要略低于其他语言。

## 4. C#

C# 是微软开发的面向对象编程语言，C# 有许多优于 C++ 的特性，比起 C++，C# 在面向对象方面做得更好。C# 用于 Web 开发比 C++ 好，而底层开发却远不如 C++，C# 需要运行在 .NET Framework 之下，有一定的跨平台性。

**提示** .NET Framework 实际上就是个虚拟机，但很遗憾的是 .NET Framework 只有 Windows 系列操作系统中才有，这也在很大程度上影响了 C# 的覆盖面。

## 5. PHP

PHP 是一种内嵌在 HTML 中的服务器端脚本语言，它是开放源代码的。经过历代开发者的辛

勤劳动, PHP 从 1.0 版的只有一个简单的网站计数器和留言本, 发展到 PHP 5 强大的面向对象功能。在开源项目如火如荼的发展形势下, PHP 应该还会吸引大批的开发者参与进来。

不过 PHP 由于一些线程安全问题、缺少标准框架和其他商业原因等(不要认为开源的就是彻底免费的), PHP 在开发一些大型复杂的项目时就有些捉襟见肘, 因此 PHP 比较适合于中小型的项目开发。

## 6. (Visual) Basic

Basic 最初是为初学者设计的一门语言, 后来被计算机厂商(尤其是微软)不断改进。Windows 操作系统问世后, Visual Basic 逐渐流行, 它的最大特点就是易用, 可以方便地连接数据库, 快速建立 Windows 应用程序和企业级程序。

同样是由于 Visual Basic 的易用性, 很多人认为不应该将其作为初学者的入门语言, 不利于初学者学到基础的编程模式和结构及养成良好的编程习惯。而且用 Visual Basic 开发的程序只能运行在微软的 Windows 平台之下, 也在一定程度上制约了其用武之地。

## 7. Perl

Perl 是一种具有动态特性的脚本语言, 同时支持泛型变量等灵活的特性。由于 Perl 借鉴了其他编程语言的很多语法, 使得 Perl 比较容易学习。Perl 主要应用在 UNIX 平台下的程序开发和 PHP 的开发中, 具有一定的跨平台性能。

## 8. Python

Python 是一门比较年轻的语言, 在 2004~2005 年非常流行, 随后发展平稳。其最大的特点就是可以和 Java、C++ 等语言很好地结合在一起, 而且只要添加某种模块, 就可以实现相应的功能。Python 主要应用于多媒体处理、网络编程等方面。

## 9. JavaScript

JavaScript 不同于 Perl 等脚本语言, JavaScript 是运行在客户端的脚本语言。其简单易学, 但是要想用好必须下大工夫学习。尽管安全性差, JavaScript 还是靠着其跨平台性、灵活性高等特点成为最受欢迎的脚本语言, 尤其是在如火如荼的 AJAX 技术渐渐成为大众新宠之后。



很多初学者容易混淆 JavaScript 与 Java, 其实二者之间主要的联系就是名称中有四个英文字母相同, 其他的如应用领域、运行平台等都有很大的区别。Java 可以开发从单片机嵌入式程序到大型企业级应用, 而 JavaScript 主要是开发嵌入在浏览器中运行的应用程序。

## 10. Ruby

Ruby 和 Python 以及 Perl 比较类似, 最大的不同之处在于 Ruby 里所有都是对象(而在 Java 里字面常量被封装之前并不是对象), 而且 Ruby 比后两者都年轻。Ruby 有一个基于 MVC 模式的框架 ROR (Ruby On Rails), 在解决中小型应用时快速简洁。

Ruby 由日本人发明, 所以网络上的资源并不是很多。关于 Ruby 是否能流行于大型商务应用, 笔者持观望态度。

## 11. Delphi

Delphi 是由 Borland 公司开发的，它并不是一门语言，而是个开发环境，主要使用 Object Pascal 语言。很多人熟悉的“熊猫烧香”就是用 Delphi 开发的。除了开发 Windows 下的应用程序，Delphi 还可以应用在 Linux 平台下。



**显示** Linux 平台下的 Delphi 不称之为 Delphi，而称为 Kylix。不过由于 Windows 与 Linux 操作系统平台所提供的接口不同，并不是所有用 Delphi 开发的软件都可以无缝迁移到 Kylix 中。

## 12. Pascal

同 Basic 一样，Pascal 一开始也是为教学而设计的。因此 Pascal 是一门很好的入门语言，用于数值计算也非常适合，也是很多计算机类大赛的参赛语言，同时很多名牌大学也是将其作为入门教学语言。Pascal 语言产生了很多版本，Delphi 采用的就是其中的 Object Pascal。

“怎么样，蔡佳娃？看了这么多编程语言的介绍，心里有没有什么打算啦？”

“师兄，越看越迷糊啦！”

“没关系，可以好好想想，我说的可能不全，你可以在网上再搜一搜资料。”

“OK！”

选择一门编程语言，只是入门的途径。过分依赖编程语言，只会让自己成为代码高手，而不是开发大牛，要知道编程语言只是一种工具，更重要的是编程思想。

### 1.1.4 你说我容易吗

前面介绍了外行人眼中 IT 人的印象，其实家家有本难念的经，在貌似风光的职业背后，IT 人也有 IT 人的辛酸，也有一些不足为外人道的苦衷。

下面向读者介绍几个 IT 人不容易的方面：

- 水涨船高的薪水
- 脑力负荷重
- 技术更新快
- 压力大

“蔡佳娃，上回你讲的是你对 IT 这个行业的看法，这回换我这个行内人给你吐吐我们的苦水了。”

“是吗？那我得好好听听哪！”

#### 1. 水涨船高的薪水

“就像我之前说的一样，IT 从业人员大都分布在东部沿海地区，而这些地区恰恰也是中国物价比较高的地区。所以就薪水来说，北京的年薪十万和石家庄的年薪十万是大大不同的概念。”

“听你这么一说，果然有道理。”

“不仅如此呢，除去物价，必须考虑到现在中国越炒越高的房地产，想想中国房价最贵的地方，温州、上海、北京、杭州、深圳、广州……那些 IT 中心城市，哪个房价不是排在前头啊。只能说不幸，每个 IT 精英所向往的工作地同时也是中国最寸土寸金的地方。”

“师兄，我错了，原来IT人也不容易啊！”

业内心声：薪高一尺，价高一丈，所以要想在IT界精英的聚集地安个家，对于每个IT人都是个不小的难题。但是，一个真正的IT人凭借着自己的努力，总是可以实现这个目标的。当然需要的时间就只能因人而异了。

## 2. 脑力负荷重

“还有啊，虽然IT人挣高薪挣得有些假，可是工作负荷重却是毫无疑问的。”

“此话怎讲呢？”

“IT几乎是个纯脑力劳动的产业，对于细节的把握非常重要，因为一个小小的编程错误可以造成数百万甚至上千万的损失。本来劳动强度就够大，还要不时加班熬夜，很多人步入IT行业一两年后，由于太累扛不住，转行的人也不少。”

“哎，的确是这样哈，IT人赚的每一分钱都是智慧和死亡脑细胞的结晶啊。”

业内心声：除了IT，大概找不出第二个对智力依赖如此高的服务行业了，严谨求实的就是这个行业的标准。再加上熬夜和加班，开发人员的脑子承受着不一般的负荷。开发人员一边为算法的优化费尽心思，一边又在为找bug排除故障抓破头皮。

## 3. 技术更新快

“蔡佳娃，你觉得我们IT行业发展快不快啊？”

“那是相当快啊！日新月异都不行，得是‘分’新‘时’异。”

“发展快对谁都好，就是对我们开发人员不好，这就意味着我们必须抓紧时间学习新技术，在IT行业，新技术就是逆水，不学则退啊。好多公司每年都投入很多财力、物力来搞培训。”

大家都知道庄子在《庄子·养生主》中曾经说过：“吾生也有涯，而知也无涯”，但是大家几乎全都忘了后面几句：“以有涯随无涯，殆已；已而为知者，殆而已矣”。全文的意思是：我们的生命是有限的，而知识是无穷尽的，用我们有限的生命去探求无限的知识，肯定不会有好结果的。

然而IT这个光鲜亮丽的行业却必须是一个需要不断学习的行业。IT仅仅在其诞生的几十年中就已经发展到如日中天的地步，靠得不正是迅猛的技术更新速度！光看硬件产业，在著名的摩尔定律的带领下，计算机配件的性能和价格变脸般地更新着每个人的生活节奏。

所以如果你在满负荷的工作之后没有抓紧时间学习新技术，那么结局大概就像《多收了三五斗》里那样，只是多混了三五年经验，结果反倒贬值了。

## 4. 压力大

“说到这，我如果跟你说IT行业的压力很大，你还有意见吗？”

“呵呵，绝对没有了，看来IT这个东西，不是人人都能扛下去的啊。”

“是啊，不过如果你能扛下去，做到了高层，或者是技术元老，站到技术的最前沿，你就真算是修成正果了啊！”

“师兄。希望你赶快变成高层啊！”

“嗯那！呵呵。”



**业内心声：**概括来讲，IT 人享受着平均高薪，普遍高压，追赶着先进技术，做着脑力劳动。这些可能不是每个 IT 人现在正在做的，但却是每个 IT 人曾经做过的。总而言之，IT 是个勇敢者的游戏，没胆量的人是玩不到头的。

### 1.1.5 我挨踢我骄傲

尽管 IT 人存在着前面介绍的种种难处，每年还是有很多人涌入 IT 这个职场淘金，而已在 IT 很多年的老手也很少会去转行。既然是这样，那么干 IT 这一行绝对还是有很多让人流连忘返之处的。

- 外人的眼光
- 高成就感

#### 1. 外人的眼光

一项工作体不体面都是外人说了算，很多外行人都一般地认为从事软件开发的人脑子都特别好使，这让很多开发人员听了，就算工作很苦，压力很大，还是觉得很值。

诚然，现在一些 IT 公司员工猝死的事件也让外界对这个行业的玩命程度肃然起敬，但是 IT 这个流行职业还是以其超赞的智商劳动和前沿的技术手段惊喜着每个人。累是累，但是 IT 人吃的都是智商做的饭，香！

#### 2. 高成就感

用自己的聪明才智赚钱，肯定是很欣慰的，况且从事的还是对很多人讲都讲不明白的高新技术。身为 IT 人，看着闯入人们生活的种种 IT 产品，那种通晓内幕的沾沾自喜是很难有其他事物能够比拟的。

设想一下，你是淘宝网后台系统的开发人员，你看到别人在淘宝上买东西，你会很自豪。因为你通晓这个用来买卖东西平台的来龙去脉，了解整个业务流程，甚至还知道一些别人不知道的 bug 或窍门。此时，再看着他在电脑上一步一步按着你定下的规矩操作，内心的窃喜是不言而喻的。

就算很少会有人听你讲解这其中的奥妙，因为在他们看来这不过只是“茴香豆”的四种写法，但是身为 IT 人，感受那种真理掌握在少数人手里的感觉，妙不可言。

## 1.2 当今主流公司的企业文化

除了高校和科研院所，IT 人才基本上都是集中在各个公司的。而各 IT 公司因为文化背景、行业领域、公司规模等方面的差异，也存在着不同的企业文化。IT 人在入行之前研究清楚什么样的公司唱什么样的歌，根据自己的特点选择合适的公司，防止自己“入对了行却选错了铺子”是很有必要的。

### 1.2.1 欧美企业的特色文化

计算机技术起源于美国，当然欧洲也贡献了大量的优秀数学家和计算机学者，所以欧美（当然也包括日本）的 IT 技术是全球领先的，欧美公司大概是每个 IT 精英都想去一展身手的地方吧。因此，欧美公司的特色文化必须先搞清楚。

“蔡佳娃，上回让你看了编程语言的排行，现在做决定了吗？”

“不急嘛，师兄。你再给我多介绍介绍呗。”

“OK，今天我来跟你说说现如今各个公司的企业文化。”

“企业文化？有必要吗？甭管什么公司直接进去干不就行了吗？”

“哎，你总是逼我鄙视你的无知啊。不同的公司当然有不同的文化，听我讲完你就大概明白了。”

“呵呵，那师兄你赶紧给我讲讲吧。”

“先说说欧美公司的企业文化，欧美公司的技术含量是最高的，而且欧美公司是每个 IT 精英都最希望去的理想企业。”

“这是为什么呢？高薪吧？”

“不全是，主要是欧美公司比较人性化，工作氛围十分自由开放，而且上下级各个阶层交流都非常直接，有些地方还会实行弹性工作制，就是尽最大限度让员工感到安逸，进而很自愿、很自觉、很自发地努力工作，证明自我。”

“的确啊，看来欧美公司的确是我们的天堂啊！”

“不过，一分钱一分货，没有真本事，欧美公司的门槛我们是很难挤进去的哦！”

“千真万确，不过我很期待自己能有一天得偿所愿，呵呵。”

欧美公司（主要是美国的公司）比较根本的原则就是以人为本。欧美公司认为一个企业最重要的军事力量是人才，而人才是需要被尊重、保护的，所以那些企业鼓励员工学习培训，努力让员工感到适应，从而心无旁骛。

欧美公司重视人才，也体现在其招聘人才、构建最适合工作的环境、员工福利方面，不过就像故事里讲的那样，多大饭量多大碗，虽然欧美公司招聘人才不拘一格，但是想要进去，恐怕还是需要自己头上顶上了“精英”的光环才行。

这里需要提一下德国的 IT 企业文化，德国素来是科技实力比较高，学术氛围比较浓厚的，高斯、哥德巴赫、莱布尼茨等伟大的头脑都来自德国。德国的企业严谨踏实，纪律严格，跟中国比起来可以说毫无人情味儿，不过也不完全是坏处，只需要搞好技术，可以不必研究烦琐的人际关系，这也是很多 IT 人所不擅长的一项。

最后，以微软为例，简要说明一下欧美 IT 企业的基本团队模式，如图 1-2 所示。



图 1-2 微软项目团队模式图

首先从图 1-2 中可以看出，欧美企业内部的各个团队角色是平等的，没有森严的等级制度。

下面来介绍一下这个团队模式中的各个角色及其职能。

- **Project Management:** 项目经理，负责整个项目的总筹，掌握资金、人员等。
- **Development:** 开发人员，负责按照项目需求进行项目开发。
- **Testing:** 测试人员，和开发人员合作进行项目测试。
- **Logistics Management:** 质量管理，项目测试无误后，内部或外部试用，验证项目软件的可用性及易用性。同时，也包括其他一些与项目相关的后勤工作。
- **User Education:** 用户培训，负责培训用户项目软件的使用法。
- **Product Management:** 产品经理，他是开发人员和用户之间的纽带，综合用户的需求，将项目的设计方案交给开发人员，或者将项目人员的观点反馈给用户。

另外微软项目管理主要是协调三个因素的平衡：

- **Resources**（资源）
- **Features**（功能特性）
- **Schedule**（时间表）

上述三个因素也称为黄金三角，如图 1-3 所示。

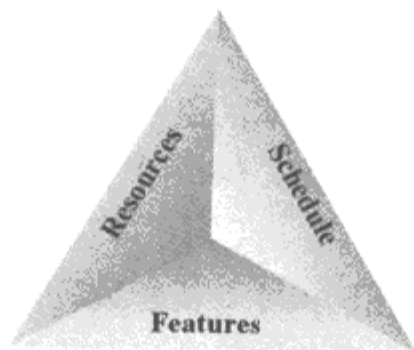


图 1-3 项目管理的黄金三角

其中资源主要包括人力和财力，在实际开发中需要根据客户的要求，将 3 个方面的因素进行调整。比如客户要求进度，那么在功能实现上可能需要妥协一些，资源要求也会比较高。

### 1.2.2 日韩企业的工作模式

日本和韩国的 IT 企业在中国也有不小的规模，尤其是日本和韩国（主要是日本）的软件外包业务占中国的软件外包业务总量的比重很大，所以那些日韩外包的接包企业也或多或少地受到了日韩企业文化的影响。

“刚才我们讨论了欧美企业的文化，现在我们来讲讲日本和韩国的企业文化。”

“日本和韩国的企业也很不错吧？应该比国内的好吧？”

“别急，听我说完就知道了。每个人的看法都不一样。下面我只谈谈我的看法。”

“嗯，你说吧，我洗耳恭听。”

“我听我一个朋友说日韩企业的管理方式就是吸血鬼式的管理，千辛万苦地榨取员工的每一滴血，千方百计让员工拼命。在那里，不加班是不合群的表现。”

“啊，这么残酷啊。”

“这就是日韩企业的最大特色，不过这并不能算是坏处，你想想看二战过后日本和韩国为何经过几十年的发展就迅速成长为经济发达的国家，就是靠着这种玩命的干劲。你肯定听说过日本是

世界上工作压力最大的国家吧。”

“嗯，不过这样应该还不错吧，至少可以让公司和个人都得到发展呢。”

“光是这样当然还可以，不过日本企业的等级制度比较严格，和中国的企业有些类似，所以办个什么事情比较麻烦。”

“嗯，这样一来还不如咱中国的公司好呢。至少是为自己人拼命。”

“呵呵，你看，愤青了吧。其实日韩的IT企业主要讲究的就是团结，跟欧美公司最大的不同就是日韩企业讲究团体实力，而欧美公司更加侧重打天才战术。就好比美国的NBA或其他体育项目喜欢打明星战术，而不像日韩的团队体育精神。”

其实故事里关于日本IT公司有一个方面没有提到，那就是日本公司对员工的福利特别好，也算是对员工玩命工作的一种补偿吧。

韩国的IT公司和日本差不多，不过韩国的公司对员工的企业文化培养比较重视，真正工作之前会接受比较全面的培训，再有就是韩国IT公司的纪律也是非常严格的，这一点和日本公司差不多，所以有时会感觉很受限制。

其次，韩国的升职制度也比较成问题，所以在韩企升到高层比较困难，不过韩国公司的坚强干劲还是很值得大家学习的。

### 1.2.3 中资企业的传统特色

既然是中国的IT行业，存在最多的公司必然是中国自己的公司。中国的IT公司在IT大发展的三十年间逐步改进，受到欧美和日韩等国家的公司文化影响，再结合中国传统企业文化的积淀，逐步形成了比较中庸的企业文化，这也比较符合中国传统的儒家文化。

“讲到了最后，终于该提提我们自己国家的IT企业文化了。”

“就是啊，我也纳闷呢。为什么不先说我们国家的呢？”

“我这么做是有原因的。欧美和日韩的企业文化差异比较大，而中国的企业文化比较中庸，前两者的特点兼而有之，所以我们放到最后说。”

“哦，那师兄你说说怎么个中庸之道吧？”

“虽然中国的官本位文化比较浓厚，但是在IT这个比较前卫的行业，中国的公司受欧美的影响还是比较大的，还是有一些以人为本的特征的。”

“是吗？这样可太好了。”

“我有一个朋友就任职于一家中国的IT企业。他们那里就很有人情味儿，上级和下级的关系都很好，纪律也不是特别严格，也很少加班，但是很多事情还是领导说了算。”

“这也不错哈，工作环境应该比日韩企业要好一些。”

“其实有些地方也和日韩企业比较相像，比如也会有很多篮球、足球之类的社团，公司也会找时间组织员工去旅游啊什么的。”

“哦，看来中国的IT企业的确比较中庸，比不上欧美公司的激情，也没有日韩公司的严格。”

跟欧美其他国家不一样，我国的IT发展时间不长，而IT在短短的历史中诞生了大量的技术。因此大量技术更类似是“涌入”的，再加上欧美等其他国家公司的进入，对我国的IT企业造成了

多方面的影响，所以显得中国的 IT 企业看起来就像是欧美和本国传统企业的混合体。

同样是由于上述原因，中国的中小型 IT 企业远多于大型 IT 企业，这也就使得中国 IT 行业中的团队模式一般规模不会很大，分工也不是很细致，比较类似于欧美的自由模式，但又很像日韩的官僚等级，其实官僚主义并不是什么特征，任何比较大型称得上“帝国”的公司都会滋生这种东西，比如 IBM。

中国的 IT 企业文化还在不断地探索自己的道路，既接受着外国企业的文化冲击，又摸索着自己的道路。国外的文化不一定全有用，不可单纯地搞“拿来主义”，中国的企业需要自己的特色。比如中国的员工私下关系都很好，而外国的企业很大程度上都是为了工作才交际。

#### 1.2.4 两种不同的软件外包方式

在 1.2.2 节中，本书向读者介绍了中国软件外包企业的行情。软件外包产业将会是中国软件出口的一项重头武器，所以本节将继续向读者介绍现今存在于中国的两种不同的软件外包方式：人员流动和项目流动。

“蔡佳娃啊，还记得上次我们说到的 IT 软件外包产业吗？”

“记得记得，主要面向日韩欧美的嘛。”

“嗯，不错。今天我们就来说说中国软件外包产业的两种不同方式。”

“这还有分别啊？不就是人家让干啥就干啥呗，完事了给钱就行了吗？”

“此言差矣，哪有这么简单啊，老是这么简单地想问题迟早会吃亏的。中国的软件外包产业分为两种方式：人员流动和项目流动。”

“那师兄你给我讲讲吧。”

##### 1. 人员流动

人员流动，即发包方向接包方要求开发人员的租用，接包方将开发人员送到发包方进行工作，实际上就是外包的内容是人。被租出去的开发人员便会去发包方公司上班，而薪水和其他福利政策则由接包方负责发放和实施。

外包出去的人员并不一定都是开发人员，很多公司并不缺少开发人员，而是缺少软件测试人员，所以外包软件测试人员的也很多。

“我们先谈人员流动，人员流动其实说白了就是‘人口贩子’。比如说 A 公司打算开发一个软件项目，需要外包，但它不希望把项目拿给人家去做，所以它与 B 公司签约，从 B 公司借来一些开发人员，让他们参与到开发的过程中。”

“那些被借来的开发人员到底算是哪边的人呢？”

“可以说生是 A 公司的人，死是 B 公司的鬼吧，呵呵。也就是说上班去 A 公司，做的工作也是在为 A 公司创造财富，但是薪水由 B 公司发，出了什么合同或者福利方面的问题也是去找 B 公司，从这一方面也说明，IT 人才的流动性是很大的。”

“哦，这样也很不错啊，有种双重间谍的味道，呵呵。”

“不错个鬼啊，就是因为有了这层关系，给很多假冒的猎头公司或者经营外包业务的公司带来了一些可乘之机，用来行骗。”



“行骗？那是怎么做的啊？”

“你比如说B公司招聘过来一些人，‘贩卖’到A公司去，而A公司自然不会为那些人的薪水、医保、保险等方面操心，而B公司又总是闪烁其词，一般会发发工资，但是其他的就不管了，这样员工一旦出了什么事情，A公司和B公司双方踢皮球，最后受苦的还是员工本身。”

“哎，看来做什么都不能太急，必须研究无误了才能做决定，就像签合同。”

“是啊，‘You can never be too careful.’”

中国的软件外包企业中采用这种外包方式的还有很多，如比较出名的如软通动力、文思创新等。这种情况下一般发包方（即故事中的A公司）是一家规模比较大的公司，当其遇到比较大的项目时便会找像故事里B公司这样的公司来解决人员缺口的问题。

接到项目再去找人开发，这是一种很常见的公司运营模式，很多大的公司如IBM的CSDL就是这样，正式员工只有50%，其他大部分都是“贩卖”过来的。所以被贩卖过去后如果表现上佳，还会被发包方聘为正式员工，那样就算是“从良”了。

人员流动是针对很多公司的这种节约成本的运营模式而产生的。较项目流动还是有一定先进性的，不过正像故事里说的那样，再遇到人员流动的外包情况时，一定要弄清楚自己的位置，保证自己出了什么事情都有人来管，切勿上当受骗。

## 2. 项目流动

项目流动，即发包方直接把项目的供求说明告知接包方，接包方收到后组织自己的开发人员（这时候的主旨人员就有可能用到人员流动）进行项目开发，然后在完成开发和测试后，将项目交给发包方。整个过程双方只有项目的交流，并没有人事上的走动。

“好了，介绍完人员流动，接下来再讲一讲项目流动。”

“嗯，是不是A公司把项目交给B公司做，然后A再把钱给B公司啊？”

“很正确，这个应该比较符合你们对软件外包的基本想法吧？”

“嗯，这个听起来就很像外包了。不过我觉得这种方式应该没有人员流动好吧？”

“从某一方面上讲，大概是这样，因为如果作为开发人员被外包到别的公司，你就会学到很多在原公司所没有的东西，同时也有可能为自己的职业生涯创造另一条路。我有一个朋友刚刚大学毕业居然在应聘的时候打败了有三年工作经验的一名老手，而那名老手之前就是就业于主营这种业务的外包公司的。”

“嗯，师兄所言极是，我想说的就是这个意思！”

“不过不能一概而论，有些人就是喜欢安定，喜欢重复熟悉的事情。而项目流动方式就可以提供这些保障，而且项目流动方式并不是不好的职业生涯。”

项目流动是中国软件外包的另一种运作方式，这种外包目前在对日韩的一些外包业务上还是很流行，比如大连的一些外包公司都是采用的这种方式。

不过这种外包对开发人员的前途一般不好，因为有的时候发包方或者接包方已经把要开发的项目做了十分精细的分工，交到开发人员手里，开发人员只需要按照特别详细的设计做填空题就行了，设计类，编写函数，一切都是制定好的规则，有时会精确到把参数和返回值都规定好了。

所以在这种方式的开发中，开发人员并不能对项目有一个整体的把握，并且也失去了思考的机会，脑子也就停止了高层的运转。这样时间长了，在思维等其他方面自然也就落后于其他同行，不过这种方式倒是很适合初入行的新手锻炼磨合。

### 1.2.5 加入什么样的公司

“怎么样，蔡佳娃？听了这么多介绍，心里有点谱了吧？”

“嗯，听师兄你这么一说，我想了想，还是优先要追求一下欧美的 IT 公司。追不到也没关系，至少知道自己不行了，可以继续再追，呵呵。”

“呵呵，精神可嘉啊。其实就算追不到，可以先去国企历练历练，因为中国的 IT 一直都是以欧美为模板发展的。”

“对，我就是这么个打算，最好不要去那些日韩的企业，虽然那些地方福利好，精神足，干劲强，不过我想还是让自己优先学习些新的理念和技术，最好还是少做些拼命的工作，毕竟身体是革命的第一本钱嘛。”

“你很会为自己着想啊。不过你的想法也没有坏处，如果我建议你的话，大概也是这么个策略，毕竟是搞 IT 的，就得向技术最先进的方向看齐嘛。”

“嗯，欧美的技术先进，混进去至少能蹭个脸熟呢。”

故事中的建议内容大概只能作为参考，毕竟随着全球化的趋势越来越迅速，欧美、日韩和中国的 IT 企业都在慢慢地融合和交流。很多企业虽然还保留着自己的传统，但都在互相取长补短。真正的 IT 行业该走向何方，或许将由正在阅读本书的读者来决定。

不过请读者谨记，光嘴上说去哪是没用的，没有真本事是哪里都不能去的。有才能走遍天下，无才是寸步难行的。

## 1.3 散兵游勇还是团队作战

真正的软件开发行业并不像一些人想的那样，所有的项目由很多团队开发，也不像另一些人想的那样，全靠一个天才创造。两种情况都是存在的，关键是要在竞争如此激烈的 IT 行业做好这两种人，让自己有备无患。

### 1.3.1 哪样多一些

“师兄，我要问问你啊，要是真正进到了一家 IT 公司，一般自己干的时间多还是和团队集体干活的时间多啊？”

“你是想问团队合作开发的时候多还是自己独立的时候多吧？”

“嗯，我就是这个意思，想知道自己以后要是真的进了公司，对哪方面要求高些。”

“这个是完全说不准的，而且是完全没有必要说准的，你考虑的太多啦，蔡佳娃。”

“那我总得做好准备吧？”

“你应该是全做准备。这两种方式你都要会，都能做得来，这样才行。”

“哦，我还以为做好一方面就行了呢。”

“不对，谁也说不好你会用到哪个。IT 这个行业的变数比较大。你加入的公司可能因为你跳槽等缘故改变，你所在的公司接受的项目规模也总在变，所以你必须两种情况都应付得来。”

“那至少哪个比较多，哪个比较少呢？”

“哎，你还是没有听明白，没有人去统计这个的。可以这么说，欧美公司（包括面向欧美的软件外包公司）因为接手的项目比较大，所以需要团队的开发会多一些；而中国的中小型 IT 公司比较多，它们要求个人的单独作战能力必须要强一些。”

前一节介绍过，中国的中小型 IT 企业比较多，团队规模一般比较小，所接到的项目不可能很大，所以更多的情况是会把项目分成很大几块，然后交给团队开发，因为项目分割得很开，所以团队中每个成员所做的工作更多地像是一个独立的项目。

中国的一些大公司也有自己独特的经营模式和理念，不过在团队模式上相比欧美还是有所欠缺的，比较突出的一个就是中国的团队模式一般比较随意，分工比较模糊，规模也不太确定，不像欧美公司那样分工明细。

刚刚走出校门的大学生，自己单干的能力或多或少肯定是有的，团队意识一般就很少有人有所体会。所以对这方面还是有所顾虑的，就像故事中的主人公，其实团队开发并不是什么高妙的东西，当项目经理把任务分配给每个人的时候，你接下来要做的就是自己来做，和自己开发是没有太大区别的。

### 1.3.2 团队和单兵

“师兄啊，团队协作的时候应该怎么做才好呢，究竟团队意识是什么啊？”

“团队意识这个概念很多时候都被放大了。其实你只要在团队中做对自己最有利的东西，同时也做对团队最有利的东西就好。”

“那自己单兵作战的能力呢？”

“不管你将来面临什么样的工作，这个能力是你必须要有的，单兵作战能力对整个项目的统筹和把握要求较高，对整个项目从需求分析到测试都必须全局关注和执行。”

“哦，不过如果独立开发的能力很强，会不会不合群呢，让人觉得没有团队意识，影响整个团队项目的进度和性能呢？”

“不光是独立开发能力强的人不合群，其他能力不行的人有时也是没有团队意识的。我们公司以前有一个员工，并不是很厉害，不过却很喜欢指出别人的错误。我们一起开发一个项目，他对自己的代码不仔细研究，问题多多，却总是喜欢找别人代码上的缺陷，结果惹得团队里都没好气，使得项目的开发过程也不是很愉快，差点耽误了开发进度呢。”

“那如果一个开发人员的技术过于厉害，会不会对整个项目的开发造成些坏的影响啊？”

“不会啊，技术好了怎么还会变成缺点了呢，技术越好对整个项目的把握越好，越会朝着团队的目标前进，也许会有些不合群，没有团队意识吧。如果这样的话就比较危险了。”

“对，团队意识说小是小，说大是大！”

“嗯，说得很有道理。能力提高下工夫就行，做个对团队有利的人却不容易啊。”

像是在故事里说的，能力再强也要有团队意识，很多高手就是团队意识较差，使得自己瑜不掩瑕，反倒竞争力变弱。高手的无团队意识主要表现在如下两个方面：

- “诸葛亮”型的开发高手，作为 Team Leader 或项目负责人，高手事必躬亲，总觉得别人做的自己不放心，大大小小的事情都要操持。长期下来，自己每次做项目都干了几几乎全部的工作，而手下几乎没得到什么成长。小项目还顶得住，某一天接到一个大项目，高手也就慌了，因为再牛的高手也需要时间，而手下人却都眼巴巴地望着他，心想：这下你一个人做不来了吧。
- 一个高手在团队里做队员，如果他是个安分守己，按时完成任务的低调高手，这样还算好，如果高手是个不甘寂寞的人，表现欲比较强，总是不当地批评别人开发中的错误，没人愿意被自己同级的人骂，高手如果总是这样压制别人，肯定影响到队伍的团结，这样整个团队的战斗力就被大大减弱了。

在欧美的团队模式里，比如在前面提到的微软的团队模式中，分工是比较明确的。越是大的项目，越需要这样，这样各个部分可以各司其职，减少其耦合度，比如结合图 1-2，开发人员只需要和产品经理和测试人员互相沟通即可。

在日韩企业的团队中，项目分工也是很好，只是过于明细，尤其是外包业务，所以对于开发人员来说只需要做做“体力劳动”即可，而且这样的话团队之间的联系交流程度相对比较小，这种团队不能说是具有独立开发的能力，因为每个团队成员对整个项目并没有很好的把握。

### 1.3.3 不要停止思考

“蔡佳娃，我们继续刚才的话题，我想要说的是，不管你以后到了公司做什么，都必须记住，不要停止思考，这才是开发人员的 Golden Rule。”

“不要停止思考？”

“是啊，作为一名开发人员，你得明白自己是个脑力劳动者，如果有一天你发现自己的身体比心智还要疲惫，你要小心，是否自己已经变为一个披着脑力劳动者外衣的体力劳动者了。”

“啊，这都有可能啊？”

“是的，所以不管是在团队还是自己独立开发，照搬自己或前人既有的东西可以，但是不要因此麻痹了自己的神经，遇到什么事情总想着从哪搬来凑数。要让自己的脑子时刻处于思想新东西的活跃状态，你才能感受到创新的快乐。”



做脑力劳动时间长了，形成一套固有的模式，不再善于思考是很多开发人员犯过的错误。这对职业生涯的继续良好发展有很坏的影响，读者要多多注意。

## 1.4 这条路大家都是怎么走的

从编程菜鸟变成业界大牛，这或许是每个走出校门或者踏入职场的人最根本切实的希望。不过对于很多新手来说，这条荣耀之路有时只是一个概念，并没有明确清晰的计划或蓝图。本节将帮助读者把这条路看清。

### 1.4.1 职位和待遇是怎么升的

“师兄啊，其实我最想咨询你的事情是这样的，一般来讲，一个菜鸟进入 IT 这个行业，如何一步步走上去，把自己搞得很成功啊？”

“这种成功之路一般来讲每个人都不完全一样，大都是能力和机遇的双重作用吧。机遇这个因素很重要，很多 IT 风云人物都是抓住了 IT 这个神奇行业所体现出来的机遇，才一夜之间或者短期之内将自己从默默无闻变成鹤起成名。”

“嗯，机遇这个东西很难遇到啊。”

“不是很难遇到，机遇不是公交车可以等到，机遇都是嗅出来的。我们的世界不是缺少机遇，而是缺少发现机遇的眼睛。”



如果在这里再次重复牛顿和苹果的事情，也许会对这本书的创新指数大打折扣。但不可否认的是，牛顿之所以是牛顿，就是他能从苹果看到重力，就像 IT 巨人们能从集成电路中看到未来一样。机遇是给有准备的人的，这句话永远也不会过时。

“师兄你说得很对啊，那我还是先把自己的能力好好搞上去吧。”

“正解，机遇我们有时不好把握，但能力却是实实在在的东西，可以提升。一般情况下，进入公司，升职还是要靠自己的能力。”

“那具体这个过程是怎么样的呢？师兄你描述一下吧。”

“首先你进入一家公司，肯定要从底层干起，先做开发人员，然后崭露头角之后可以做到 Team Leader，负责整个小团队，然后慢慢发展，会给你负责整个项目的开发，也就是类似项目经理，但是只是技术上的负责和领导。”

“哦，那然后呢？”

“然后就是你要考虑的问题了，你是更喜欢做技术还是更喜欢做管理，喜欢技术就是负责整个公司的技术发展，比如技术主管；喜欢管理就逐步走向项目经理或市场等领域。当然做技术主管必须也会管理，项目经理也必须懂技术，只是侧重点有所不同罢了。”

“哦，我稍微明朗一些了。”

“其实根本不用把这个研究这么透的，这些东西你在步入职场慢慢提升自己的时候都会逐渐了解的。”

“哎，我就是想提前感受感受呗，幻想一下自己的美好未来激励一下自己，呵呵。”

其实，蔡佳娃的这个想法非常正常，一位 IT 成功人士回忆起自己的创业经历时曾经说过，当我和几位创业伙伴在北京的地下室里艰难地熬过三年中的每一个日日夜夜的时候，让我们坚持下去的，不是什么宏伟的念头，而是简单地幻想着成功之后的甜蜜和喜悦。

所以，白日做梦不是什么坏习惯，关键是做完梦之后，是苦笑一下了之，还是努力试着让它实现。

“你这个想法倒是不错，IT 行业有时的工作是很让人泄气的，我们有时也这么想，不管会不会变成现实，至少帮助自己把当下熬了过去。”

“嗯，师兄说得是。那薪水的巅峰之路是怎样的啊？”

“这个当然和你的职位息息相关了啊。再说不同的公司规模不一样，同样的职位也许会有不同的工资呢。研究这个没有多大用处。”

“哦，师兄那你讲讲你的成就之路吧。”

“嘿嘿，谈不上。其实你们以后去找工作，大多也是去中国的公司，或是外企在中国的分公司，



中国公司里面的职位制度不是很明显，更多的是体现在薪水上的。我就把我当时进 IBM 的经历讲给你听听吧。”（见图 1-4）

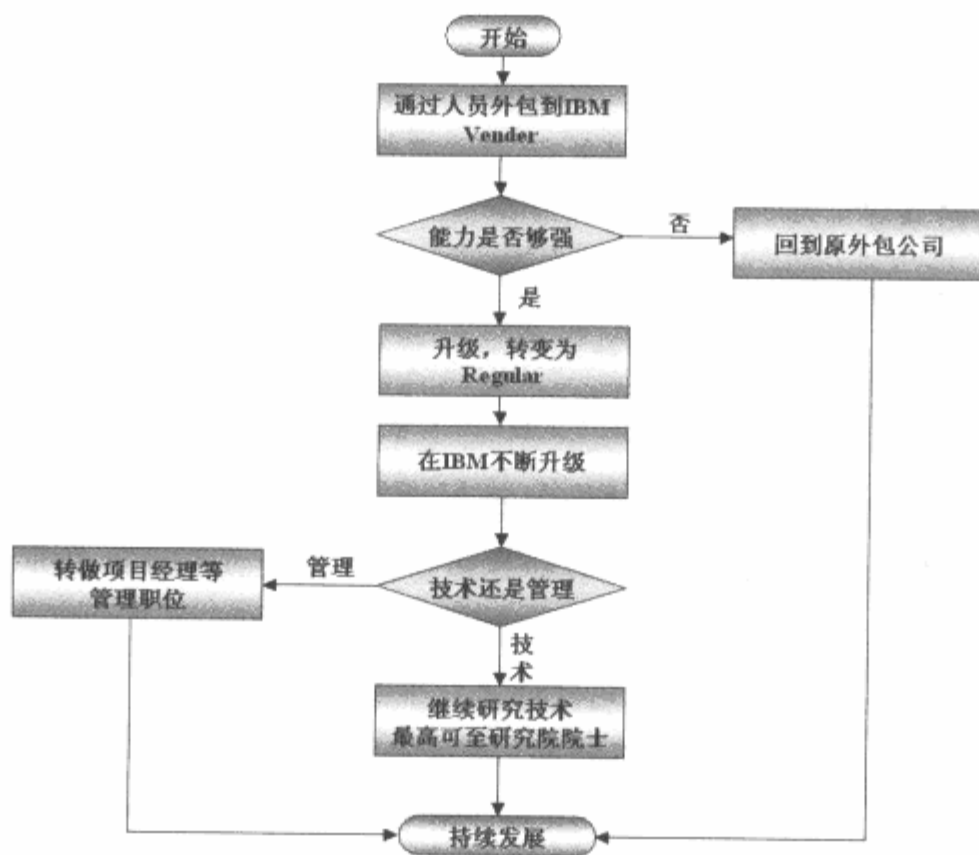


图 1-4 IBM 成功流程图

图 1-4 是 IBM 中国软件开发中心的一个简单的升职流程图。虽是跨国公司，但其实中国的 IT 公司大概也是这个流程。需要先说明一下，故事中的牛开复同学是通过软件外包的形式进入 IBM 工作的，所以图 1-4 指的也是这种形式下的职业生涯发展过程。

**提示** 这里需要提一下的是 IBM 对员工的技术能力是划分等级的，类似于柔道几段那种。分为 1 到 9 级，9 级是最高级别。

### 1. Vender 阶段

这是进入 IBM 最初的职位，由于是“贩卖”进来的，所以还不能彻底算是 IBM 的员工。但是想要进去也不是那么容易的，一般要求有一到两年的工作经验，能力必须也要符合 IBM 的招聘制度。Vender 一般的等级为 5 级或 6 级，也会有少量 7 级的。

### 2. Regular 初期阶段

项目开发完成以后，即外包合同已经结束。这时候对于 Vender 一般有两条路可走，能力比较出众的，就有可能被 IBM 聘用为正式员工，即 Regular，这时候就算是一个真正的 IBM 人了；能力并不是很出色的一般继续回到原来的公司工作。

### 3. 技术巅峰还是管理牛人

变成了 Regular 之后，作为 IBM 的正式员工，就要开始在众多的 IT 精英中求生存和发展了。Regular 继续发展，又会面临两条路可走。

- 第一就是继续做技术，经过努力让自己的技术等级不断提高，IBM 对于高技术人才政策是很好的，会提供最好的环境进行技术研究。当然这里的技术就不再只局限于某个具体的业

务项目了，一般都是一些技术发展方向、新的框架等。最高级别的技术人员是 IBM 研究院院士，这个级别很难达到，因为全亚洲也不过百人。

- 第二是走向管理层，从 Team Leader 到 Project Manager，然后慢慢做到区域总经理，一步一步升上去。最后可能会很少研究技术，主要是研究公司的内部管理、业内合作、投资的运作等关系到公司发展方向的内容。

这些升职经历都只是作为参考，虽然只有短短几百字，但是真正要做到可是需要异于常人的努力的，万丈高楼平地起，做事要有愚公移山的精神。

### 1.4.2 有干不动的时候吗

青春饭，任何有志于从事 IT 这个行业或者对其有所了解的人都会听到过这种描述，这个原先用在文艺圈的词现在又拿来吓唬 IT 人。

“师兄，有个问题我一直很不清楚，究竟我们这一行能干多久呢？很多人都跟我说这是个吃青春饭的行业。”

“不是你一个人这么问过我了，真是好事不出门，谣言传千里啊。第一个这么说的人肯定是个被 IT 行业淘汰的人。”

“是吧，我也觉得要真是这样，这还不如演艺圈呢。演艺圈都讲究实力派，老练成熟的演员有时更受欢迎呢。”

“呵呵，说得很对，所以你以后要批评所有再提这个论调的人。”

“他们是不是干不下去了所以才这么说啊？”

“并不是只有行内人才这么说的。有这种观点的人或者是个外行人，道听途说，就像你这样；或者是行内不成器的人，自己干不下去了，只好搬出这个说辞做借口。”

对于认为 IT 行业是“青春饭”的行内人来说，干不下去了一般有以下几个原因：

- 对工作无热情，进入 IT 行业只是听说能赚钱，对工资的热情远远高过工作。这样的人很快就无法忍受 IT 行业中踏实严谨的作风，没有工作热情，自然无法在竞争激烈的 IT 行业生存下去。
- 每天的工作是重复性质的，这可能是所在工作环境的缘故。如此长期下去，积极性也会慢慢消磨殆尽；又或者是个人能力达不到，职位也无法得到升迁，自然慢慢懈怠下来。
- 健康问题，这种情况比较少见，不过这种情况也比较情有可原。IT 行业毕竟是比较累的，如果就职的公司喜欢让员工加班，随着年龄的增长，自然健康状况会变糟，最后只好提前离开这个行业。

“蔡佳娃，下面给你讲讲计算机程序之母老而弥坚，终生奋斗在 IT 行业最前沿的故事吧！”

“好，我洗耳恭听！”

“生于 1906 年的美国海军女将军格雷丝·默里·赫珀或许是判决‘青春饭’错误的最好证据。格雷丝·默里·赫珀首先是著名的数学博士，被人称做是“计算机程序员之母”、“编译器之母”、“COBOL 之母”，她是计算机创造初期第一批的程序员。”

“二战时期她在美国海军服役，负责计算机程序方面的研究，期间她为计算机程序设计的发展做出了非常大的贡献，并荣获海军将军的荣誉。退役以后它还在为海军工作，因为，海军竟然无法找到人来接替她的工作。那时，她已经是个六十多岁的老人了。一直到她去世，她都一直在做

着我们这行人认为‘青春’的东西。”

“啊，这真是我们应该学习的楷模！谢谢师兄，我会朝这个方向努力的。”

或许我国的 IT 发展年限还较短，很少会有上年纪的开发人员。美国等其他国家还是有很多研究学术或者研究技术的老当益壮的大牛的。可能很少人会做到像他们那样的声誉和地位，但是至少应该让自己陪着这个行业走到最后，实现自己的人生价值。

对于如何让自己干下去，其实并不难。

- 保持始终饱满的工作热情，很多 IT 巨人的工作热情都是让我们难以理解的，像比尔·盖茨之所以能够建造一座宏伟的微软大厦，这和他对计算机技术的热爱是分不开的。
- 通过做一些不可避免的重复性工作，通过量变达到质变，提升自己的能力素质。坚决不要做那些披着脑力劳动者外衣的体力劳动者，那样很快就会被淘汰。

希望看到这里本书已经说服了各位读者彻底与“青春饭”论调划清界限。随着时间和职位的变化，体力劳动在减少，脑力劳动、知识的积累和丰富的经验却在不断地增加。这些都是时间给予每个人无价的财富。

### 1.4.3 走的人多了，还会有路吗

鲁迅先生在《故乡》中曾经说过：“这正如地上的路；其实地上本没有路，走的人多了，也便成了路。”IT 行业从一开始的没有路，飞速发展成现在的高速公路。而 IT 界对应文豪的话，却有了另一番解释：“地上本来有路，走的人多了，也就看不见路了。”

“师兄，你说现在有想法从事这个行业的人这么多，会不会路越走越窄啊？我听说现在这个行业的竞争非常激烈啊。”

“那你说高考的千军万马过独木桥这路窄不窄？你不也过来了吗？”

“说的也是啊，可是……”

“首先批评你一下，就像上次跟你说那个‘青春饭’的问题一样，很多事情都不要道听途说。不过这次与‘青春饭’有所不同，因为竞争激烈倒是真的。”

“的确，不过干什么应该都不容易吧，师兄？”

“那是，想想看当初 IT 前辈们创业的时候竞争倒是不激烈，可是他们也没什么可利用的资源，没有前人的经验作指导，没有先进的仪器搞发明，但是他们却在艰苦中彰显卓越，为我们开辟了一个 IT 时代；而现在我们虽然竞争激烈，但是却有很多资源可以利用，有很多书籍作为指导。”

“是啊，比起创造 IT 时代的前人们，我们的确是没什么可抱怨的。”

“正解，所以无法在竞争中生存，只是自己不够厉害，失败应该从自己找原因。”

“师兄你说得对，面对竞争我们必须有勇气去赢。”

“真正的高手越是在艰苦不利的环境中，越能表现出超高的战斗力和超强的抗打击能力。”

鲁迅先生还说过：真正的猛士，敢于直面惨淡的人生。目前竞争激烈的 IT 行业还谈不上“惨淡”，所以面对着竞争，任何职场新人都应该拿出本该燃烧的激情和初生牛犊的胆量，在 IT 的竞技场里面都能做个猛士。

不管怎么说，IT 都是一种职业，既然作为谋生手段就会有改行的可能。除非有高到青天的智商和厚如大地热情（如果真是这样的话也早已经是蜚声内外的大学者了），总有感到疲倦或厌倦的时候。升职、转行、创业都可能是 IT 职业生涯的最终出路。

“师兄，继续前天的问题，假如万一我真的在这一行干不下去了，或者厌倦了，我还能有什么出路呢？”

“哎呀，你想得还挺周全。的确，当初和我一起进公司的同事，有些也都已经转行了，留下来的只有一多半。不过人各有志，去留肝胆两昆仑嘛。”

“那他们最后都还混得可以吧？”

“那我就不清楚了，有的上了一层台阶，有的跌了一阶。但都算转行了，对吧。”

在IT行业中转行不干的原因很多，最常见的就是觉得无法再有更高的成就（当然不是已经达到顶峰）了，或者总是混不出头，再或者就是要自己单干当老板等。铁打的IT职场，流水的人才，这也算是一种良性循环吧。

总结起来，在IT行业做到中途离席转行的，一般出路如下：

- 进入高校做老师，选择这一条出路大概是受够了原公司不间断的熬夜加班，不想让自己趁年轻拿命换钱，到老了拿钱换命。不过这条出路也是比较困难的一种，因为现在高校老师这个职位的安逸是很多人都想追求的境界，竞争绝对不亚于IT行业。所以想要达到这种要求，硕士、博士文凭不是万能的，但是没有也是万万不能的。
- 努力爬到管理层，这条路比较正统，因为到了管理层，至少冥思苦想的繁重脑力劳动会大大减少，严格意义上讲应该不算转行。但是的确会将开发人员从那些压力中解脱出来，当然这条出路的难易程度因个人的能力而异。
- 转而做培训，IT行业火爆，竞争激烈，于是这种提升竞争力的培训项目也就有了市场。开发人员转去做培训应该是最简单的一种方式。不过这种出路表面上和当高校教师类似，实则不能保证稳定性，但是退而求其次，况且培训也是待遇不错的选择。
- 自己单干，这种出路应该是最让人向往的，也是最有挑战性的。自己出来单干的原因应该很多，或者是嫌自己挣得钱配不上自己的实力，或者是与公司高层意见相左，或者是想自由自在，或者是有自己的新idea等。自己出来单干成功的例子有很多，但是还有更多我们不知道的失败经历。不管怎么说，自己开公司当老板都是很有魄力的。

世界上第一张软盘的设计者艾伦·舒加特就是一个非常好的“单飞”榜样。艾伦·舒加特在IBM服务了18年，最后毅然地选择离开。舒加特先是在一家公司做高级工程师，随后于1973年创办了舒加特联合公司，专门生产软盘驱动器，但一年后他被自己公司的风险投资家解雇了。

1976年，蛰伏三年的舒加特王者归来，先是宣布研制成功5.25寸软盘，之后与人合伙成立了舒加特技术公司，舒加特在这次创业中大展身手，他发明的微机系统接口（SCSI）和研制成功的能与软盘兼容的硬盘驱动器，都成为后来行内的标准。

1985年，舒加特将公司改名为希捷，谁料之后公司便陷入了困境，希捷公司的客户（包括最大的客户IBM）都与希捷公司中断了业务，之后舒加特的合伙人也与其分道扬镳，成立了另外一家公司，与舒加特进行了长达10年的殊死竞争。

1996年，舒加特收购了最大的竞争对手，即自己曾经的合伙人创办的公司，至此宣告希捷成为世界硬盘厂商的巨人。不过舒加特注定不能安逸地享受既有的成功，1998年68岁的他又被自己亲手创建的希捷公司强行辞退。被解雇后，舒加特和别人又创办了一家风险投资公司，专门扶持和培养小企业。

舒加特的创业史可谓跌宕起伏。在他几十年的创业生涯中，在遭受重重挫折的时候，舒加特仍然像个勇士般地战斗，不仅在领导公司和创造财富上都取得了重大的成功，还因为自己的天才创造赢得了行内人的尊敬。

- 转而做生意，或者从事其他毫不相干的行业。这种出路就很难说了，因为彻底和自己之前从事的活动无关了，也就无从评论了。

## 1.5 大公司，小公司

归根结底，所有求职者最后还是会进入公司上班，公司按照前面小节里讲的分为欧美的、日韩的、中国的、传统的、外包的等。但是最普遍的一种分法就是大公司和小公司，简单来说食量不一样，吃的多少和吃的方法也就不尽相同。

### 1.5.1 大公司爱专才

大公司的概念应该是员工达到 500 人以上规模的公司。大公司里面的团队模式比较正规，因为要管理好数百的员工，除了思想教育，就是纪律了。

“蔡佳娃，还有一个问题得跟你说说，到你找工作的时候，不仅要看公司的国籍、性质，也要看它们的规模大小。”

“师兄的意思是大公司给的钱多，小公司给的钱少？”

“不是啦，这都不一定，小公司由于成长较快，需求人才度高，也许会提供比大公司更高的待遇，而大公司都是见过世面的人了，或许你的才能到那里并不会得到重视的。最后再声明一遍，直接决定待遇的，永远是自己各方面的能力。”

“嗯，师兄我下次再也不提钱了。”

“好，我们继续说，大公司的机构肯定是比较庞大的，官僚气息也会比较严重，我指的不仅是中国的大公司，国外的大公司也是有官僚主义现象的。”

“嗯，这个应该不怕，毕竟长这么大，官什么样咱也都见识过。”

“呵呵，下面讲讲大公司的开发团队，这是和你关系比较大的。大公司因为人多，所以管理一个项目也比较层次分明，每个员工的任务也比较明确。”

“嗯，这应该就是大公司的好处吧。”

“不错，大公司对于员工的要求有一点就是会的技术可以不多，但是要专而深。大公司人才很多，你会多了却没一样技压群雄的也没用，我们公司的团队里面就是，小组里每个人都精通不同的技术，加到一起就很厉害，因为这样我们的技术领域既没有盲点，又没有重合。”

“嗯，的确是这样，一招鲜，吃遍天。”

大公司的项目管理结构大致如图 1-5 所示。

一般情况下大公司接手的项目规模也必须要大，因为巨人肯定不能靠只麻雀果腹。既然项目规模大，分解开来，每一部分仍然很复杂，所以还会逐步细分。项目分到最后，每个部分所涉及的知识面已经非常狭窄了，所以只需要在这方面精通的人来开发即可。



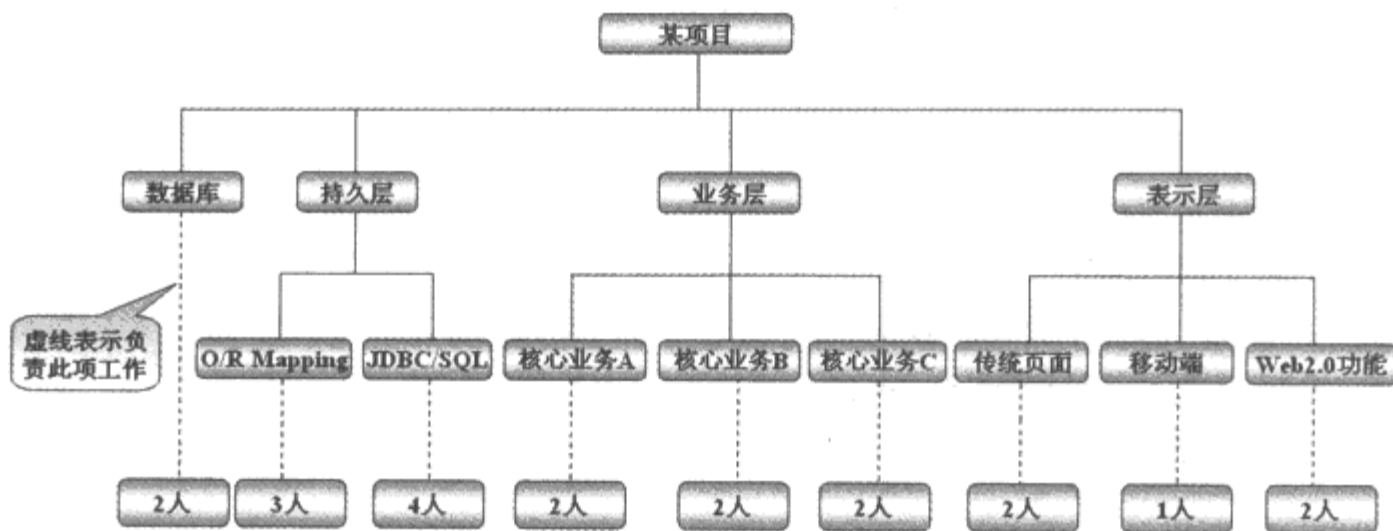


图 1-5 大公司项目管理结构示意图

大公司肯定不会缺少技术人员，但是肯定缺少专精于某项技术的人才。所以只是技术很全面的人去大公司工作会很难过，虽然很多技术都懂，但是总有人比自己更精通。时间长了，自己什么也学不到，反而让上级认为地主家也没有余粮养闲人。

### 1.5.2 小公司爱多面手

小公司的规模一般比较小，员工在 50~100 人左右。小公司一般没有能力为每种技术都配备一个领域高手，所以小公司的职员虽不能“万事通”，也差不多算是“百事可”了。

“师兄，那你说说去小公司是怎么个状况呢？”

“小公司肯定不会那么人才济济啦，所以小公司更喜欢那些熟悉很多技术的多面手，用大而全的知识来满足客户的需求。”

“看来大小公司的需求还真是不一样呀。”

“小公司的项目管理模式一般比较随意，因为人也不多，有时候整个公司就只有一个开发团队，所以很多事情大家都是身兼多职。”

“这种方式其实也挺锻炼人的。”

“是啊，这种方式让每个人对整个项目都有一个很好的把握，做起来更容易，和团队其他人的交流也更容易。”

“呵呵，师兄，看来大公司和小公司各有各的不同啊，孰好孰坏，不能一概而论哪。”

小公司的项目管理结构大致如图 1-6 所示。

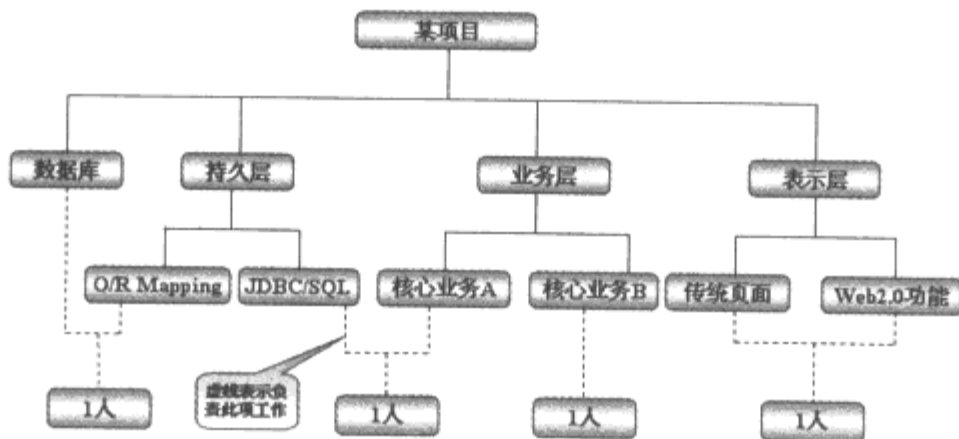


图 1-6 小公司项目管理结构示意图

虽然小公司接手的项目不算大，但是麻雀虽小，五脏俱全。就像演员们常说的：没有小角色，只有小演员。小公司的项目由于规模、资金、开发周期的限制，不可能寻找专精的开发人才来进行开发，而进行项目开发的人员就必须身兼数职，多劳多得一些了。

其实，存在于大公司和小公司之间的还有大量的中型公司，这些公司既追求像大公司那样的专才，也需要像小公司那样的多面手。

正像故事里说的那样，公司规模大小并不是判断自己待遇高低的标准。进了大公司可能道路好走些，但是也很难在那里做到鹤立鸡群，而且门槛也会很高。而小公司门槛低些，上升空间可能会更快，但是技术深度比不上大公司，公司理念和文化可能也不会有很深的底蕴。

## 1.6 本章小结

本章向读者揭秘了现今 IT 行业的种种“内幕”，旨在让广大读者在读过本书后对风光无限、风头正劲的 IT 行业能有一个全面的了解。当然了，知己知彼才能百战不殆，了解行业更应该了解自己。让自己变得有用和不可替代才是王道，因为哪家的地主都不会有余粮的。

# 第 2 章 雾里看花——职场误区

通过前面一章的行业揭秘，读者应该对 IT 这个行业有了一定的了解。但就像海洋拥有宝藏的同时也拥有漩涡暗礁一样，IT 行业也存在着一些误区。“触礁是小，沉船是大”，因此在步入 IT 这个充满成功与激情的世界时，看穿职场的种种误区就显得很有必要了。

## 2.1 到底差不差钱

都说谈钱会很俗，这应该是本书至少第二次提到钱了。说实话谈钱其实并不一定俗，在经济环境下让自己的财富增加是件无可非议的事情。不过在 IT 这个行业中，尤其是新入行的人，如果把钱看得太重，绝对不是一件好事。

误区：眼中的 IT 更多的是个淘金地，而非一个刺激新奇的技术天堂。处在这种误区的人，不一定技术很差，而是薪水在其心中的位置远远超过了对技术的渴望。

分析：或许因为是自己这么多年来一直在学校接受教育，总是伸手要钱，让自己不自觉地背上了沉重的心理债务。又或许是因为接触的社会让自己变得急功近利，于是毕业之后迫切需要寻找到一个高薪的工作。

蔡佳娃就是处在这种误区里的人。因为来自农村，可能对钱的感悟更深刻一些，或许是想对辛辛苦苦供自己读书的父母有个实实在在的交待，所以在咨询牛开复的时候，总是时不时就扯到钱上来。

“师兄，我这样是不是不太好啊？我想到一个职位的时候，第一个念头总是挣多少钱。”

“有你这种想法的人还真不少，不过你算是少数的几个能自己看出问题来的。”

“那师兄你说我该怎么去掉这个念头啊？”

“其实想到钱并不是什么坏事，关键是不要把 IT 这个行业仅仅作为一个挣钱的途径而加入进来。把钱看得太重，就把自己看轻了。”

### 2.1.1 大家都是个什么身价

根治这种症状的第一步，就是让其彻底看清大家到底都是什么样的身价。不过这服药可不好熬出来，薪水这个问题，从哪个角度讲可能都不一样。横向可以分为不同学历、不同技术、不同公司、不同职位、不同资历等，纵向又可以分为不同的时期，这着实是个统计学的大难题。

所幸，这一步并不是最重要的，可以本着保证治愈的原则在统计细节上稍微放宽一些。

#### 1. 起步价

每一个步入职场的新人都必须从试用期这个起点出发，但是就像出租车也会因为城市、车的档次、油价行情等因素定制不同的起步价一样，IT 这个行业的起步价也千差万别。这里统一进行一个假设，即所有人都是能力不低，至少是以优异成绩毕业的职场新人。

“蔡佳娃，为了让你不再陷入钱的误区，我先给你讲讲现在 IT 人的薪水情况吧。好让你彻底搞清楚这里到底能淘到多少金。”

“嗯，还是师兄你最了解我。”

“首先谈谈大家的起步价，即试用期的薪水。由于学历、公司、地域的不同，起步价还是有很大差别的。为了方便，假设大家都是顺利毕业，成绩优秀，能力比较强吧。”

“啊，这么高的标准啊！”

“不然怎么样？假设大家都是不学无术、混吃等死的乌合之众吗？IT 行业永远追求的是最高的标准，IT 人也不能这么不思进取呀。”

“是是，那师兄你说吧？”

“首先是学历的不同，硕士研究生一般比本科生的入行薪水高 40% 或者更多。有些公司的某些职位是只招收硕士以上研究生的。”

学历对于一些 IT 方向的在校大学生来说是个比较纠结的话题，往上考吧，怕进去学几年出来跟不上形势掉了队；直接顶着学士学位冲吧，又怕人家以文凭取人把自己埋没了。其实很多时候 IT 公司招聘最看重的还是个人能力，所以不管是什么学历，能力才是最先要保证的。

“啊，看来多读几年书真不是白读的啊。”

“除了学历，大公司和小公司的试用期也是不一样的，大公司门槛理所应当地高，所以对于招进来新员工的能力还是比较有信心的，所以放心大胆地给钱。而小公司就不一样了，它们一般不会冒这个险，所以试用期工资就较低一些。”

“是啊，毕竟是大公司，做事肯定要有派嘛。”

“也不能太绝对，一般小公司并没有太雄厚的资金，某一职位的薪水可能要比大公司的同样职位少很多。不过，有些小公司规模不大，但是正处在快速上升的阶段，十分需求人才，所以薪水不一定会比大公司少很多，甚至会更高。”

“是呀，所有大公司不都是从小公司成长起来的嘛。”

“正解，所以小公司也是不能全部小觑的。”

需要说明的是，虽然能力有高低，进入 IT 职场的初期，基本上大家的试用期工资都差不多。这是 IT 行业收入两极分化最不严重的时期，正所谓站在同一个起跑线上。当然还要考虑公司所在城市的差异，因为不同城市的物价指数和消费水平是迥然不同的。

参考价位：2000 ~ 4000 元。其实不应该设置上限，因为 IT 本来就不是那种中规中矩的行业，一切奇迹都有可能发生，就看天才怎么出招了。

## 2. 上升价

上升价是指在新员工逐渐熟悉了工作环境后，随着工作时间的增加以及经验和能力的提升，在几年之内给自己带来的薪水上升空间。如果起步价是公司给定的话，上升价就完全靠自己的努力了。这里又该提一提 IT 行业的“一切皆有可能”了，因为真正的差距将在这里拉开。

“师兄，新员工的薪水我大概有些了解了，那么新员工的薪水能涨到多少呢？”

“一般员工在工作了一到两年之后，薪水会提高 6 成或者更多。当然这只是一个平均概念。需

要考虑的是所从事的技术是否具有很高的应用价值以及所在的城市，另外大公司和小公司的增长幅度也是不完全一样的。”

“哦，那是怎么个不一样法啊？”

“大公司的薪水是普遍较高，而且薪水制度和福利制度比较完善，比如 IBM 的新员工本科毕业试用期后每月大概基本工资 5000+，但是完成的任务越多，加成也越大。公司还会为员工设立补助账户，租房补贴和年假、探亲假等制度也都很健全。而小公司薪水可能会相对低一些，但最主要的还是福利、休假制度不很健全，这些其实也是待遇的一部分啊！”

“是呀，有时候工资是不能完全代表待遇的。”

“当然了，上升期能上升多少，最重要的还是看自己的个人能力。”

“说得也是，这个时候不管是人是妖，都是原形毕露的时候了。”

“没错，上升期的身价能增加多少就全靠你自己的能力了。我认识的一个师兄就挺牛，工作了两年多一点，年薪就达到了十八万，而且在房价猛于虎的广州买了新房。”

“哇塞，那他肯定非常厉害吧，技术大牛？”

“也不全是，他大学的时候学习可不怎么样，差点就毕不了业。”

“那他技术方面肯定很拉风吧？”

“怎么说呢。他这个人对于核心 Java 的领悟绝对不一般，不过 Java EE 他并没有深入学习，只是在毕业前研究了一下 SSH 框架和当时并不流行的 Java FX 技术。之后就大大方方地到广州找到了一份工作，五个月后，就跟我们说已经做到 Team Leader 了。”

“的确是不一般啊，看来薪水的提升幅度不仅和个人能力有关，恰当地把握新技术的趋势对于获得高薪也很重要啊。”

上升价时期可是决定一个人能否在 IT 职场淘金成功的重要阶段，上升期始于起步价之后，终于稳定价之前。上升价时期一般 IT 员工都会随着技术能力的熟练和经验的增加将自己的薪水提高到一个不错的水平。

值得注意的是女性 IT 员工的身价上升速度一般要快于男性，或许并不应该感到奇怪，因为世界上最初的程序员都是女的，只是后来计算机程序变得越来越流行和商业化，男性程序员才慢慢地加入到这个行业中来，并且毫不留情地占领了这块多金之地。

IT 世界中存在着悬殊的差距，上升价的高低因个人能力而定，一般是在最初工资的基础上翻 1~N 倍。如果碰到的是一个天才和一个庸才，那这两者的上升空间可能会相差 10 倍都不止，这就是 IT 世界，一个充满差距却又无比公平的世界。

参考价位：4500 元~不限。

### 3. 稳定价

稳定价比较简单，就是以个人能力达到事业巅峰之后平缓发展时期的身价，或是已经工作 10 年以上时的身价。稳定价再往下发展就是自己拉大旗干，不过这种方式的身价不好衡量，所以不做考量。

“师兄啊，那我的工资上升到什么时候算是个头啊？”

“总有一个临界点，过了那个点，你的工资就不会再有大的涨幅或跌幅了。”



“那是个什么样的临界点啊？”

“比较平淡的情况下，就是工作 10 年以后。除了极少的大器晚成之外，10 年的工作会让你的事业趋于平稳。因为如果你在这个行业中 10 年都没办法让自己的工资达到一个非常高的层次，你基本上在今后的 10 年里也不会有比较大的动静了。”

“有道理啊，张爱玲不也说过嘛，出名要趁早。”

“呵呵，差不多。稳定价应该算是薪水飞涨的最后一站了。这个时期薪水一般都是论年薪了，而且一提年薪一般都是 10 万以上了。”


“嗯，我希望我的稳定期来得晚一些，好让我的薪水多上升一些。”

“傻啊，上升价可不是靠上升期的长短决定的。好比一条直线，决定它上面点的  $y$  值可不只有  $x$  值，斜率也很重要啊。”

稳定价的时候已经到了个人事业发展的平缓阶段了，一般职位都在项目主管、经理之上了，或者已经成为某一个技术领域的专家了，而这个时期的薪水一般只和专业领域及职位有关了。

这里需要指出的是每个人的人生追求都不一样，对成功的定义也就不同。因此稳定期的定义也出现了分歧，大致分为以下三种：

- 像李开复、唐骏那样的 IT 高级管理人才，他们缔造了一个又一个的 IT 帝国，谁也不知道他们心中的火什么时候才会熄灭。总之他们肯定会锐意进取，不断创造着 IT 的神话。
- 有些人不擅长也不喜欢管理别人，就喜欢管好自己、研究技术，这些人最后会成为公司的技术总监、首席科学家或者行业内的专家。
- 深知“一分耕耘，一分收获”的道理，知道人家的高薪是忙碌打拼出来的；本着“知足常乐”的原则，自己拿个 5000~8000 元的工资也就满足了，这并不是不求进取。其实知道如何收手，有时也是很难做到的。

 **提示** 就像本节中出现最多的一个词“一般”一样，每个人都不是别人的复制品，每个人都有自己的独特地方。对于富有冒险精神的 IT 人来说，本节的薪水介绍仅仅能作为一个参考。自己的价值，最终还是需要自己来创造。

### 2.1.2 给自己估个好价

看了大家的身价，想必读者心里也开始盘算自己的身价应该定位为多少了。准确地给自己一个定位，不仅会让自己找到合适的职位，更能让自己以最好的姿势起跑。当然，不要太早给自己确定最终的定位，现在的你还有很多准备升值的空间与时间。

“师兄，看了各个时期 IT 人的身价，我发现有必要为自己也定一个合理的价钱了，呵呵。”

“是呀，知己知彼，方能百战不殆嘛。了解自己要像了解敌人那么透彻才行。”

“那应该如何为自己的能力打分呢？”

“很多情况下，我们并不是放在超市里贴个标签写着价格的商品，而更像是菜市场里被人挑来挑去还不时威胁‘一块二卖不卖，不卖我走了’的南瓜或是土豆。我们中很少有人能做到让人蜂拥抢购的程度，能做到展览拍卖的就更加寥寥无几了。”

“说得也对，现在的 IT 行业竞争实在是不小啊。”

“所以这个时候对自己的评估并不是我‘会’做什么，而是我‘能够’做什么。两者是不同的，‘会’做什么是对自己现在能力的定位，而‘能够’做什么是对自己将来能力的肯定。要对自己未来的能力有所预见。”

“那到底估价是为了什么啊？”

“研究了这么多关于职场的问题，想必你能为自己的未来勾画出一份蓝图吧。估价，从某种意义上来说，更是对自己的一种规划。你为自己估的什么价位，你自然会让自己时时刻刻的表现都符合自己对未来的规划。”

“嗯，说得也是啊。”

上述都是对还没有步入真实职场的人而言的，因为他们还有为升值而进行准备的充分时间。一旦真正到了求职的时候，就只能是结合当前的行业形势，结合自己所学专业在市场上的流程度，结合自己的能力水平来给自己估价了。那时，如果之前的准备不充分，职场不认可就只能一跌再跌了。

**提示** 读者在给自己估价的同时最好把准备工作做充分，这样才能让自己的估价得到职场的认可，才能让自己的估价成为现实。

其实有的时候给自己估价除了要看自己的能力，还要看从事的技术方向，即选用的编程语言、面向的操作系统或程序接口等。有时候由于方向选择错误，就算自己再有真才实学，也是会感慨自己英雄末路，无力回天。

### 1. 不同编程语言的收入

本书在 1.1.3 节中向读者介绍了当今 IT 行业编程语言平台的人员分布，下面继续向读者介绍不同编程语言平台的收入分布。

“蔡佳娃，还记得我们之前讨论过的 IT 行业编程语言的流行程度分布吗？”

“嗯，记得呢。就是各个编程语言的排行吧。”

“对，现在就根据每个语言方向看看它们的财富含量。”

“不是三百六十行，行行出状元吗？”

“话是这么说，不过我们现在讨论的主要是面向大众的跟薪水有关的行情问题，而你的状元想法或许只对于少数创造时势的英雄管用吧。”

“哦，那师兄你来说说吧。”

根据编程语言平台薪水分布图(图 2-1)可以看出，从事不同编程语言的工资分布有如下特点：

- 一般来说，越流行的语言，其相应的薪水也会越高，这个道理应该很容易理解。一门语言流行起来，研究的人就会增多，比如 Java 和 C/C++ 等，自然会将这门语言发扬光大，使其越来越完善。这样就扩大了其应用领域，自然收入也会升上来。
- 有些语言虽然流行度不够，但或许是新生语言（如 C# 诞生不过六七年的时间），又或许流行度不高（就像图中未列出的 Python、Delphi、Pascal 等），这些语言或者发展很快，或者在某一些方面的应用有其他语言无法比拟的优势，所以薪水仍然很高。
- 有些编程语言虽然流行度很高（如 VB 等），但是由于主要功能或效率的问题，处理的是

小型应用，并没有在一些大型企业级开发中得到青睐，所以薪水就自然无法和其他语言媲美了。

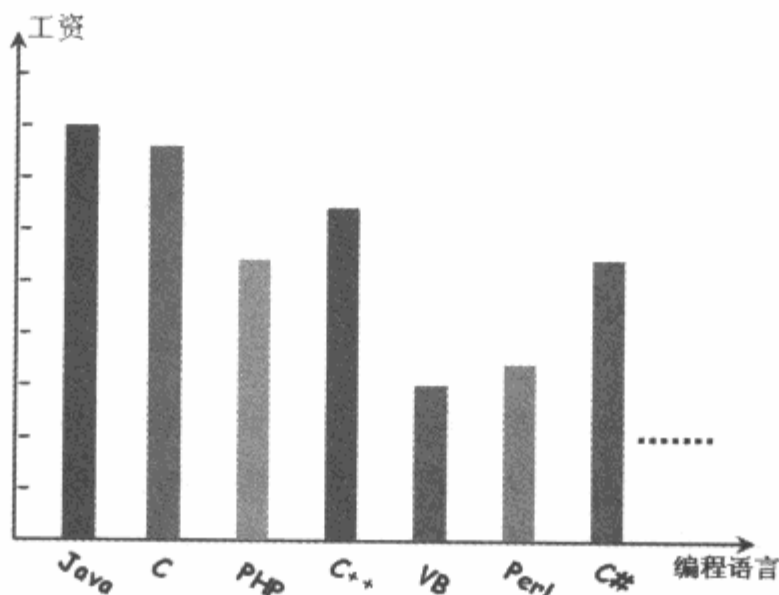


图 2-1 编程语言平台薪水分布图

图中的编程语言收入排行只是相对性的总体比较，因为毕竟鸡窝里也可以飞出金凤凰。同时，这里的薪水情况也只是作为参考，随着时间的推移也可能会发生很大的变化。

## 2. 面向不同操作系统收入

面向不同的操作系统进行开发，其收入也会有很大的不同。不过一般来讲，选用的编程语言在一定程度上决定了要面向的操作系统平台，比如 C#、VB 目前只能用在 Windows 平台下，而 Java 可以跨操作系统平台进行开发，Perl、PHP 等脚本语言也具有很大的跨平台特性，而 C/C++ 可以面向不同操作系统平台进行开发。

总体来讲，根据中国 IT 行业的现状，面向 Linux 和 Solaris 等系列操作系统的开发要比 Windows 平台的收入高一些。因为前者大部分是用来解决一些高端的大型企业级应用，而 Windows 因为是从 PC 起家，所以在面向中小型的项目应用时有很大的优势。

有多大本事，就有多大气。老话说得好：“没有精钢钻，就别揽瓷器活”。给自己估价并不仅仅是让自己更好地卖出去，更重要的是能更好地实现自己的价值。

### 2.1.3 先挣的是资本，后挣的是钱

赌场上有句话：“先赢的是纸，后赢的是钱”，这句话的意思就是赌博要沉得住气，不要在意开始时的输赢，赢在最后才算是真赢。如果把这个道理告诉那些一味追求眼前高薪的 IT 人身上，或许会对其有所帮助。

“师兄，你先是给我说了说 IT 这个行业到底能淘到多少金，又告诉我应该对自己有个精确的定位，并且对未来有个清晰明朗的规划。那么，在选择职业的时候，究竟对于薪水应该如何看待呢？”

“总体的原则是在进入职场初期，以积累成长为主，以追求高薪为辅。IT 是个成长极快的行业，金钱永远只是飞速发展的技术的附属产物，那些追求高薪的人都是本末倒置了。单纯追求眼前高薪，等 IT 行情一变，自己的金山也就沉入海底了。”

“所以，师兄你的意思就是在初次求职的时候不要太注重薪水，要选择那些自己上升空间大一些的公司喽。”

“对啊，要知道先挣的是资本，后挣的才是钱。努力把自己从菜市场的南瓜土豆变成展览会上的拍卖品，才是对自己最大的尊重，也是对钱的尊重。”

“钻石恒久远”这句广告词用来形容 IT 行业实不为过，但是后一句“一颗永流传”就很不恰当了。或许真的存在可以一劳永逸的事情，但应该不会太多地在 IT 这个行业内发生，在 IT 行业注重积累资本永远是正确的做法。

或许那句“授人以鱼不如授人以渔”用在这里是最恰当不过了，有再多的鱼都不如会捕鱼，有钱不一定有再挣钱的能力，有挣钱的能力才会永远有鱼吃。

## 2.2 谁给我解决户口问题

户口这个问题，或许有些读者并不是很了解，找工作会和户口有关系吗？有的读者或许已有所耳闻，但仍不解户口这个老实巴交的东西为何会在求职路上横刀立马。

**误区：**对户口问题处理不当，不是因为当地户口怕被拒绝不敢去求职，就是求职时张口闭口要求单位解决其户口问题。认为户口和学历能力一样重要，甚至更重要。

**分析：**或许是受了前辈人言传身教的影响，或许是经历过户口的难题已经对户口这个问题伤透了心，总之在求职的过程中，遇到户口问题就如临大敌一般。处在这种误区的求职者，大都对于户口问题并不了解，只是道听途说，其实户口这个政策问题国家每年都在不断改进。

### 2.2.1 讲讲户口的故事

户口，就是用来标识一个人的身份的。户口主要分为集体户口和居住户口，集体户口是指户口关系挂在所在单位，如果离职或者单位倒闭，就不太好处理；而居住户口是根据居住所在地而建立的，只要不主动迁移，居住户口是永久存在的。

户口问题主要就是指获得某一城市的居住户口很困难，按说这个问题不应该牵扯到职场里面。随着人才的频繁流动，尤其是在中国的一些大城市（如北京、上海等地），使得上述地区的居住户口的获得变得困难重重。

“师兄，你现在在北京工作，户口是不是北京的啊？”

“不是啊，怎么啦？”

“那师兄你不赶紧置办置办啊？你现在可是外来户口啊。”

“这个东西需要着急吗？你想得太多了吧？”

“怎么会呢？户口问题不是很重要吗？师兄你现在不是北京户口，当心以后评职称、结婚啥的有麻烦哦。”

“这就是你对户口的想法啊？看来我的师弟对 IT 这个行业的错误观点还真不少啊！”

“啊？这还算错误啊？户口不是很让人头痛的问题吗？”

“是，也不是。还是听我给你慢慢道来吧。”

“那师兄你赶紧说吧，给我扫扫盲。”

“首先给你讲讲过去的故事吧。几年前，户口或者真的是个大问题，当时因为北京、上海等大城市就业机会多，所以很多高级人才、海归人才、务工人员都涌入了这些繁华的大城市，慢慢地也就带来了管理和治安方面的问题。”

“是啊，人多了就是不好管。结果呢？”

“比如你拿了一把糖果，结果一千个人冲向你，你会不会把糖果捂得严严实实呢？结果就是这些大城市把户口政策卡得很严，人们想把户口落在那些大城市就很难很难了。”

户口政策卡得严的另外一个原因和城市的发展规划有一定的关系，曾经有一段时间这些大城市并没有意识到引进人才的重要性，主要还是偏重于引进投资。对那些回国创业的‘海归’和其他满腔热血的高级人才并不支持，有的时候还会频频刁难。

不过最近几年随着社会经济的不断发展，各地也越来越重视人才的引进，相应地也推出了很多人性化的户口政策，如上海的居住证制度等，后面的章节会详细介绍，这里读者不必过分担忧。

“师兄，户口这么难落实，岂不是很让人伤心哪。”

“是啊，所以那个时候户口是个抢手的东西啊。因为很多福利和社会待遇都是只对有本地户口的人才有效的。没有当地户口做很多事情都深受限制，所以很多人因为没有户口而没办法把家人接过来，也有很多人无法让自己的孩子在自己工作的地方上学。”

“这就是职场的户口问题吧？”

“还不算是，由于户口不能忽视的重要性，很多公司在招聘员工的时候也就加上了条件，比如本地户口优先等，这就是户口的问题。想就业，有能力还不够，没户口就得歇菜。”

“是啊，真的是很残酷的一段时期。那后来呢？”

“后来，像北京、上海等这些大城市开始意识到人才对于经济发展的重要性，而且关于户口制度的不公平也怨声载道，便逐渐开始放宽户口政策，你应该不时地听说某某明星通过什么优秀人才入境计划变成香港人的新闻吧？”

“嗯，那也算是一种引进人才的策略吧？”

先把一个人的居住户口迁到某一个城市，这样对这个人而言以后不管到别的任何地方都不算是自己的地盘，都是异乡。再加上中国人对家都有比较深的依恋，所以转来转去还是会回到户口所在地的城市。因此这种引进人才的策略，相当实在、有用。

现在很多城市在人才缺乏时也会大打户口牌，这在很多时候都能取得不错的效果。当然把前些年很金贵的户口作为招揽人才的橄榄枝，也说明当下社会对人才的重视了，作为 IT 精英的我们也应该倍感欣慰了。

“呵呵，原来是这样啊，那这么说现在户口还是很好办的了？”

“看看，又犯傻。就算政策一直在开放，但永远是处于狼多肉少的局面。所以就目前来看，户口应该还是一个困扰，并且是长期困扰中国职场人的问题。”

“那不就像我一开始说的那样喽，师兄你应该抓紧把户口落实了呀。”

“我还没说完呢。户口是比较难办下来了，但这并不是说我们这些非本地户口的人在这些地方就享受不到同等的待遇了，现在我几乎是感觉不到一点不自在的地方。我说你还是不要研究这些东西了。户口这个问题现在已经退居二线了，你大可以放心地四处奔跑找工作了。”



“呵呵，那敢情好啊。我就怕到时候我这个外地户口被人拒之门外呢。”

就像《天下无贼》里葛优大叔说的那样：“21 世纪什么最重要？人才！”。所以为了让自己更加具有竞争力，北京、上海等大城市都对户籍政策实行了不同程度的开放，就是为了广纳贤才。尤其是对于 IT 这个所有人都想引领风骚的强势产业户口政策更加倾斜，因为整个 IT 产业说到底其实实实在在的资产就是人才。

随着国民经济发展所带来的频繁交流和人员流动，各地政府正在逐步将原住民和非本地居民进行无差别管理。所以在公司暂时安身工作并不是一件难事，虽然拿不到户口，但是一些类似于暂住证、居住证等过渡临时性质的户口政策还是让类似“北漂”一族的人感受到了本地人的待遇。

不过毕竟只是“暂住”，比起名正言顺的“居住”来说，还是有差异的。拿不到居住户口，在大城市要头疼的问题主要有以下两个方面。

### 1. 子女教育问题

这个话题对于刚刚步入职场的人来说应该还是比较遥远，纵观各地的“暂住”政策，基本上对于子女的义务教育，即中、小学教育，是给予与本地居民同样待遇的。

而且，从某种角度上来说，非本地户口的工作人员子女在义务教育方面享受到的好处可能比本地学生还要好一些。比如，外来子弟在享受义务教育的时候，是根据居住地就近入学。因此如果想换一个比较好的学校，重新租房搬一搬家，更新暂住证就可以了。实现“孟母三迁”，对于外地子女来说，或许更加容易。而本地人要想这样就只有买房子、迁户口才行了。

需要注意的是，这种待遇一般也只是持续到高考前。当到高考这个人生转折点的时候，就暴露出本地户口和非本地户口的差异了，不过那至少是你从业二十年后的事情了。对于一个 IT 人来说，如果二十年都没有在逐步放宽的户口政策下搞定本地户口，那么这个人也应该好好反思一下了。

### 2. 特定社会福利问题

社会福利制度也是非本地户口目前还不能完全享受到的待遇。虽然政策已经很开放和统一了，但总是要优先保障一些有居住户口居民的特定权益。拿北京来说，比较典型的社会福利制度就是经济适用房的购买，这个待遇目前还没有对非北京户口的人开放。

不过看看经济适用房的申请资格：家庭年总收入在 6 万元以下，即家庭每月总工资在 5000 元以下。对于立志于要在 IT 这个竞技场赚得一席之地的各位读者来说，经济适用房的目标，或许是没有太大价值了。试想两个人月收入不足 5000 元，还是 IT 精英吗。

## 2.2.2 各地户口政策面面观

户口问题在很多情况下都是集中在发达城市，比如北京、上海、广州等地。而户口政策是指导如何取得当地居住户口的重要指挥棒。

尽管各地的户口政策逐步开放是一个大趋势，然而就目前的情况来看，北京和上海等地的户口仍然是针插不入，水滴不进。这里向读者介绍一下两个既是 IT 中心城市，又是户口难求城市，又是房价高高在上城市的户口政策。

## 1. 北京

“蔡佳娃，说了这么多，还是给你具体讲讲各个地方的户口政策吧。”

“嗯那，师兄，好让我心里有点数。”

“先说说北京吧。北京应该是北方 IT 的最中心城市了，北京的居住户口算是很难办下来的。简单来说，花二三百万买套房，你就可以有机会名正言顺地做一个北京人了。”

“啊？这么遥远的目标啊！”

“北京的户籍管理特点就是拿到居民户口很难，但是其尽量保证拿到的和没拿到的两个层次的人都能享受到基本同样的待遇。再有一个就是对于高科技或短缺人才提供特权，如果你可以让北京和上海为了争你而大打出手，那么你就可以选户口了，呵呵。”

“哈哈，师兄，你太会开玩笑。”

目前北京的居住户口应该是最难搞定的，北京的户口政策按照传统的方式——指标发放，每年会将有限的户口指标发放给企事业单位等机构。能不能获得，就需要看这个城市需不需要你。或者是这个城市需不需要你所在的单位，以及你所在的单位需不需要你。

北京的户口政策还有一些是面向有亲人在北京想要投靠过来的情况，如娶北京 mm 或嫁北京 gg，不过这对投靠人的年龄或婚龄有一定的限制。而对于那些更多的没有投靠对象，打算赤手空拳为自己打一份天下的 IT 新人来说，要拼得一纸户口，短期内的确不太好实现。

困难指数：★★★★★

## 2. 上海

“说完京城，我们再来谈谈上海，上海的户口刚刚改革过，比较值得欣慰的是上海的户籍政策抛弃了一直以来的指标投放方式，实行条件准入制度。”

“哇，那是什么条件呢？”

“主要一个原则就是持上海的居住证，也就是暂住证在上海居住满 7 年。其他的就稍微有一些次要了，比方说参加社保及医保、缴纳个人所得税、没有受过治安处罚和不曾参加犯罪活动等。”

“俺是个守法的好公民，这些事肯定不会做的，呵呵。”

“还有一个条件就是你的专业技术必须达标，如果没有高级职业资格，至少也应该是个中级以上的技术人才，否则你还是不能拥抱上海的户口。这个户籍政策就是想办法吸引高级人才，提高上海在全国乃至国际经济中的竞争力。”

“看来上海对于户口的开放比北京要前卫啊！”

不过这个政策还是有一定限制的。每一年都有一个控制指标，只为符合条件的申请人按顺序办理，超过控制指标后的申请人延期到下一年。如果不加控制，大量上海人‘诞生’，对社会资源会造成很大的压力。

上海在放宽户籍制度、广泛吸纳贤才这一方面做得的确不错。以前户口问题大都和购买商品住房挂钩，而上海最新的户籍政策将两者彻底分离。对于那些已在上海打拼数年却仍感陌生的“沪漂”们，着实是个很大的激励。

困难指数：★★★★☆

### 2.2.3 别怕，咱有暂住证呢

各个城市的户籍政策在逐步开放，不过变化程度最大的，还是对流动户口的扶持和重视。虽然各地的户口政策不完全一样，但是所有城市大都对暂住户口非常关照。从一开始的管理人口流动，到现在的面向流动人口服务，下面就向读者介绍一下北京、上海这些大城市的暂住政策。

其实，在暂住证和居住户口之间，还有一个证叫做工作居住证，或者叫居住证。暂住证和居住证有本质的区别，前者是由公安机关颁发的，后者是由城市的人事机关颁发的，所以含金量有很大不同。居住证可是相当于“准户口本”，所以也有“绿卡”之称。

但是随着各大城市暂住证制度的优化和改进，暂住证和居住证之间的差异逐渐缩小。目前暂住证只是相对居住证离居住户口要稍微遥远一些，其他方面两者所享受的待遇是基本相同的。随着政策的改变，暂住证和居住证可能会合二为一。

#### 1. 北京

“唉，师兄，上次你跟我说了说北京和上海的户口政策，看起来不管怎么说我们这些菜鸟肯定还是以暂住证开始的是吧？那你就给我讲讲暂住证的政策吧。”

“嗯，先谈谈北京吧，北京的暂住证制度可谓源远流长，最初的暂住证是分为A、B、C三个等级，C本是最普通；B本就有些稀有；A本就更加难求了，基本上A本就跟北京人差不多了。”

“哇，那个时候肯定是困难的那段时光吧？”

“是啊，后来这个就慢慢淡化了，现在随着北京市逐步对流动人口和本地户口实行统一对待的制度，很快将不会对两类人进行区别对待了。这样一来，流动人口在北京享受到的待遇就大大提升了。”

北京的居住户口虽然是最难拿下的，但是北京的暂住证政策却是让人欣慰的。暂住在北京的流动人口，除了在前面介绍过的子女高考问题和特定的社会福利问题暂时无法解决外，其他方面均可以放心不管，安心享受在京城的美好生活。

而且，从2009年开始，暂住证也不再强制办理，一切皆自愿。这应该是对在京城默默工作的流动人口的一项最大的安慰。

#### 2. 上海

“师兄，那上海的暂住证有什么惠民政策啊？”

“上海的暂住证制度和北京就很不一样了。上海已经取消了暂住证，将其归到了居住证里面。而居住证就分为人才引进和暂住人口。”

“那这两者有什么区别啊？”

“人才引进的居住证享受的待遇基本上和上海户口的居民一样了，而属于暂住性质的居住证享受到的待遇就很有限了，社会保险、子女教育等问题都是很难解决的。”

“看来同样是拿着暂住证，还是在北京好啊。”

“是啊，上海的户籍制度放宽了，居住证制度稍微痛苦一些也还是可以理解的。”

上海居住证制度中的人才引进类应该是属于上海市的“绿卡”了。人才引进的条件并不是特别高，主要是申请人的学历必须为学士以上、所在公司的规模如注册资金在100万元以上、个人

身体健康且按章缴纳各种税费等。

IT 属于很多城市包括上海在内的紧缺专业，所以 IT 人肯定要向着人才引进类的居住证前进，这样才能彻底心无旁骛地为这个城市和自己的未来努力了。

#### 2.2.4 户口问题小结

户口其实就是人事关系和居住地的总和，前文中提到的集体户口和居住户口的概念并不一样，集体户口拥有的只是人事关系，所以要想蜕变成居住户口还需要有一个固定的居住地。

户口政策年年在变，也许在可以等到的将来，中国将不存在户籍制度，所有人不管在哪里都不会有异乡的感觉。说到引进人才，IT 是包括北京、上海在内的很多大城市比较感兴趣的行业。而高级的 IT 人才就备受追捧了，所以不管户口问题变成什么样，对于人才永远是欢迎的。

### 2.3 我们不是爱加班

加班，就像前面章节中提到的，是很多人对于 IT 这个行业的首要印象之一。随着 IT 渐渐深入到每个人的生活，加班也渐渐成为很多 IT 新人对于这个行业的误解。

**误区：**是个 IT 公司就会加班，不爱加班的开发人员不是合格的高科技人才。IT 开发人员的高薪，都是加班加出来的。

**分析：**就像所有的误区都是由于不了解造成的一样，IT 可不是个夜战之国，开发人员也不是猫头鹰。弄清楚加班的根本原因，才能让自己不再受加班之苦。

#### 2.3.1 常态加班是为何

加班，即在正常的一天工作结束后，仍然由于自愿或非自愿的原因延长工作时间，在公司或者家里继续工作。现在加班是个再普通不过的现象了，很少有人没有加过班。不过当加班已经成为一种常态，就需要谨慎研究一下了。

“师兄，进入 IT 这个行业，是不是必须得有良好的加班意识啊？”

“良好的团队意识我倒是听过，良好的加班意识是什么？”

“就是首先心理上必须对加班有正确的认识，承认加班的长期存在性，接受加班，乐于加班。”

“啊？我好像从来没听说过唉。”

“其次呢，身体上要对加班有足够的抗打击能力，身体要能承受经常性的加班，还要保证加班时的工作质量。”

“哇，你们这些没下过水的倒是对游泳姿势研究得不少啊，呵呵。”

“不对吗？师兄你加班不多吗？”

“加班可是有很多原因的，你的这种加班意识也太太无畏了。”

“啊？那具体情况是什么样的啊？”

“别急，由我来给你分析分析先。”

下面针对 IT 界的现状，向读者介绍常态加班的种种不同。

### 1. 自愿型

“常态的加班分为好多种，首先第一种就是自愿型的，就是公司没有要求加班，自己的任务也没有到非要加班不可的地步，只是这些人对于工作实在是太有热情，这种人属于‘衣带渐宽终不悔，为伊消得人憔悴’类型的。”

“啊，这种人加班应该很享受吧？”

“是啊，完全自愿的加班肯定是双方都没有怨言的。公司肯定不会阻挠，而自己又是主动地拼命，当然是一种双赢的策略。”

“不过这样的人应该不多吧？”

“是不算多，很少有人这么用心的，毕竟身体不是铁打的，再忙也要休息的。”

自愿型加班的人，一般做事比较认真投入，并不是非加不可。只是不加觉得可惜，不加就遗憾。就像玩电脑游戏，看看表，不早了，下线吧；然后又对自己说：再玩十分钟，做完这个任务就下线。十分钟后又会再有其他的借口。

自愿加班并不是什么坏习惯，只是项目任务进行到某一点，思路好不容易理得非常清楚，不想设个断点明天接着再来。如果不一鼓作气将项目一举拿下的话，等到“再而衰，三而竭”的时候，还得去走回头路，这可是比较浪费时间和精力。

不过自愿加班并不总是对公司有好处，人都不是金刚不坏之躯，精力是此消彼长的。晚上用了功，白天肯定要略显疲惫，有可能会在公司的正常业务活动中力不从心，影响正常工作。

加班痛苦指数：★☆☆

加班感受：痛并快乐着！

### 2. 主观被动型

“蔡佳娃，刚刚说的是自愿型加班，这种加班我曾经尝试过，不过只有几次，毕竟要保证第二天的正常上班。”

“是啊，要不然光做好事还不落好。”

“不过接下来的这种加班类型就不怎么美好了，那就是主观被动型加班，我可不提倡你以后这样干工作。”

“那主观被动型加班是什么样的啊？”

“主观被动型加班是公司没有要求加班，但自己也不是自愿加班。只是自己的当天任务还没有完成，所以不得不去加班了。”

“那这种加班就很不舒服了。”

“岂止是不舒服啊，首先是没办法才加班，其次加班也不一定能够出效果。这种人不像刚才说的自愿型的，自愿型的加班至少是有米可炊的巧妇，而主观被动型的加班就是无米可炊的拙妇了。”

“是啊，这种加班实在是不可取啊。”

“被动的加班效率一般都不高，结果又将工作拖到了第二天，第二天晚上自然还得加，然后一直拖，就变成了常态的加班了。”


主观被动型加班很不好，主要还是个人的工作态度和方式不正确。究其原因，有如下两点：



- 脑子懒，不懂得思考，把任务仅当做工作来对待，而不是一项挑战。总是用笨的方法来解决，而不去花点时间研究更新更快的方法提高自己的工作效率。
- 不学习新技术，用笨的方法也就算了，还用陈旧的技术。这么一来开发项目肯定如老牛拉破车，效率低下，质量还不高。

加班痛苦指数：★★★★☆

加班感受：痛苦，痛苦，很痛苦啊！

 **提示** 磨刀不误砍柴工，多花些时间思考，多研究些新技术，这不是浪费时间，是在改进自己，让自己远离这种费力不讨好的加班。

### 3. 客观被动型

“好了，现在来说说最后一种加班，客观被动型。”

“客观被动型是否也属于非自愿的加班，只是外界原因使然的对吧？比如公司强制加班？”

“嗯，差不多，这种加班一般是客观原因造成的，不只是公司的强制，有时候公司没有强制加班，但是当加班已经成为一种作风、一种公司文化的时候，就在所难免了。”

“啊？还有把加班作为潜规则的吗？”

“有啊，最典型的就是日系的公司，在那里按时下班是要遭到上下员工的鄙视的。你就是装也要装到大家陆续开始走的时候再离开公司。中国也有很多公司是这样的，别人都加班，就你一个人摇大摆回家，你自己也不舒服，只好留下来共患难。”

“哎，当加班成为一种习惯，不加也得加啊。”

“是啊，这种加班也不是非常累人，不过公司强制的加班就比较难受了。有的人完成了自己的任务，但是由于种种原因，必须还要做一些分外的事情。”

“那加班费还是有的吧。”

“钱这个东西就完全看情况啦。有的公司会算加班费，有的则不会；有的公司会体贴地放你第二天半天假，有的则不声不吭，表示明天接着早来。”

客观被动型加班的原因有很多，有一种就像故事里提到的“加班成为一种习惯”，这种加班原因很简单，就是公司的工作氛围使然。其他的客观被动型加班原因大致如下：

- 员工的问题，有些员工工作不够努力，或者是眼高手低，夸下海口而又隐瞒军情，又或者是属于主观被动型加班。总之这些员工由于自身的缺陷耽误了整个项目的进度，到最后只好让所有参与项目的其他员工共同买单。
- 公司问题，每个员工都干得很好，倒是公司管理不善，对于项目运作的规划没有做好，只好最后让员工一起受累。
- 公司没有带一个好头，管理失策，偏偏用人不善，员工也不合格。这种问题加问题的状况使得公司只能彻夜灯火通明，员工挑灯夜战了。

加班痛苦指数：★★★★☆

加班感受：虽然并非自愿，但是考虑到大家都在加班，只好随波逐流。由于不是孤军奋战，至少内心要平衡许多。而且加班也不会完全没有补贴，只是要给别人擦屁股，怨言还是有的。

### 2.3.2 你为什么加班

“说了这么多加班的事，那师兄你是为什么加班哪？”

“我还是比较欣慰的，我们公司很少会加班，就算有也不是主观被动型加班，主管分给我的任务我都完成了，只是偶尔被公司强制加班。”

“那是公司的制度不好吧？”

“倒也不是，只是总有一两个人，能力不是很出色，但总是硬着头皮扛着，主管分配完任务问大家有没有问题，他们也不说。但是不会做是改不了的，只好一边瞒着上级一边自己找方法解决，主管不知道啊，以为他们和其他人一样都在努力工作。”

“但其实他们的确在努力工作啊！”

“话是这么说，等到了最后，主管说了，大家把做的都拿过来，我们整体调试一下吧，然后他们的部分没有完成，没办法，大家只好加班让他们的进度追上我们。”

“是啊，那种加班滋味肯定不好受吧。”

“相当不好受啊，可是也没有办法，总要以团队目标为主吧。”

了解完加班的种种原因后，如果你正在饱受着加班之苦，就应该思考自己是因为什么加班了，究竟属于哪个类型。不同类型的加班滋味可是大不相同的。

- 自愿型加班，这种加班方式感觉还是蛮爽的。有些类似癫狂状态的梵高之类的艺术家，对技术的热情可以让自己在工作上表现得更加出色，更容易让自己脱颖而出。不过这种狂热的加班最好还是不要太频繁，否则身体还是会顶不住的。
- 主观被动型：这种加班其实就是哑巴吃黄连，而且就算能说也不可以说苦，因为都是自己的懒惰和不合理利用时间造成的。而且这种加班效率不高，有时候操之过急反倒会忙中出错，所以作为开发人员，这种加班现象是要严格杜绝的。
- 客观被动型：这种加班首先是被动的，而且很有可能加班的起因不在自己，自己只是来“救火”，所以怨声可能会大一些。但是大家都在努力，所以相比主观被动型加班，熬起夜来也要稍微心安理得一些。

### 2.3.3 让自己不再加班

“蔡佳娃，说到最后，你一定应该知道自己以后如果真的走上 IT 职场，该如何让自己避免常态加班了吧？”

“嗯，只是知道了不同加班的原因和后果了。那到底如何让自己不再加班呢？”

“其实很简单，说说客观原因的加班，即由于公司或其他员工造成的常态加班。如果你不能选择一个好的工作环境，只好让自己尽量接受这个事实，或者通过自己的努力去试着去改变公司的环境，实在不行就只好走人了。”

“是啊，那主观原因的加班呢？”


“主观原因的加班如果是因为自己完不成任务，那可真要下大工夫让自己的能力提高起来，否则总是这样熬夜过关，根本无法从本质上解决问题。不仅对身体不好，时间长了可能连熬夜的机会都没有了，IT 行业淘汰一个人是可以很快的。”

“的确是啊，逆水行舟，不进则退。”

“对于那些工作太积极的熬夜，我个人还是比较赞同的，我也偶尔会这样夜战。不过要注意自己的身体，不要耗光自己的精力。因为第二天上班你打瞌睡的时候，老板可不会认为是因为你昨晚熬夜了，是情有可原的。”

其实，加班并不是一件完全不好的事，有过编程经验的读者应该都有所体会，往往对一个问题进行解答，在夜深人静的时候或是睡前最容易有灵感。而且人的大脑越是到了晚上，越是冷静和清晰。

相反早上走进公司，一般需要一段时间来慢慢进入角色，而且白天会有很多琐碎的事情打扰。因此，偶尔趁着晚上环境好、脑子静可以事半功倍地进行项目开发，也不失为一种工作的乐趣。

 **提示** 正因为如此，很多大公司都有严格的规章制度规定员工在上班时间不能随便打扰其他开发人员。因为，可能由于帮助解决你的一个小问题而打断了别人的思路，别人帮你解决问题花了1分钟节省了你5分钟的时间，恢复思路花了半小时，你总共浪费了公司25分钟。

## 2.4 莫学狗熊掰棒子

狗熊掰棒子的故事相信每位读者都听过，“每一次把棒子掰下来，都放到另一个胳肢窝里，然后用那只手去掰第二个棒子，结果第一个棒子就掉了，如此反复，到了最后，狗熊就只剩下手里刚掰下来的唯一一个棒子”。其实IT行业也存在很多有着“狗熊掰棒子”作风的人。

**误区：**从学生时代就是这样，做过的大小项目或是开发的小程序，都是做完就了事，删了或者是扔到一个以后不一定能找到的地方。自己在编程中遇到了问题，问同学，问网上高手，问老师，千辛万苦把问题解决了，也不做个心得笔记，只是享受了一下解决问题的喜悦罢了。

**分析：**这种人就是典型的具有“狗熊掰棒子”作风的人，好了伤疤忘了疼，说得一点都没错。程序开发是个脑力劳动，很多时候靠灵感想出来的东西过去一些时间后便会连自己都不能理解，更何况是自己出过问题的地方了。

### 2.4.1 做过的这辈子永远都不会忘吗

“蔡佳娃，你问了我这么多问题，我来问问你，大学这三年，你写过什么小程序，做过什么小项目没有啊？”

“小项目大概是没做过，小程序倒是写过一些。”

“那你写过的东西都还在吗？删掉了吗？整理过吗？”

“整理？那东西需要整理吗？不过倒是没删，应该在电脑的某个文件夹中吧。”

“哎，那你写小程序的时候应该也会遇到一些问题吧？还记得都是些什么问题吗？”

“问题倒是遇到过，不过当时随便去网上问了问，解决了也就没再想了。”

“哎，看来我的小弟几乎把该犯的错都犯了一遍啊。”

“啊，师兄，这也不对啊？”

“相当不对啊！你想想看，做过的东西这辈子永远都不会忘吗？犯过的错改正过一次永远都不会再犯吗？”

这句话说得很对，做过的项目和犯过的错，绝对不可能这辈子都记得。做项目的时候全靠大脑思考，而思考这个活动很难像在草稿纸上演算那样留下什么痕迹，有的时候自己都无法理解仅仅是几天之前所写的代码，更不用谈这辈子不忘了。

犯过的错误要想不再犯，其实也很难。在出错或遇到问题的时候，往往四处积极寻求指点和解答，但是一旦问题解决了，往往会长舒一口气，兴奋一下，然后继续编程。而问题的症结和解决方案却被渐渐地淡忘掉了。

“师兄，不过就算记不清，在遇到的时候应该还会很快想起来吧。”

“不要对自己的记忆抱有太多的幻想了。我有一次在网上浏览帖子，发现一个 Java 初学者问的一个问题。那个问题好熟悉，我想了半天，突然想起来这个问题在多年前曾经让我备受折磨，我对当年的气急败坏至今仍印象深刻，只是我无论如何都记不起来自己是如何将那个问题放倒的了。”

“真的吗？记住了问题的模样，却忘了如何打败它的吗？”

“是真的，后来我把那个提问者的代码复制下来仔细研究了一下，终于想起来当年的解决方案了。真是不容易啊，我以为我再也想不起来了呢。”

“看来要想一辈子不忘还是不容易办到的啊！”

“不仅仅是遇到过的问题需要记住，自己曾经开发过的小程序小项目也最好都留着。”

“那些到了实际开发中还有用吗？”

自己曾经做过的小项目、小程序是很有价值的。小项目、小程序往往是对一类特定问题的解决方案，如果能够很好地保留，在再次遇到同类问题时不但可以借鉴，有时甚至可以直接修改加以使用。这样不但方便，还在很大程度上提高了工作效率。

很多小程序甚至代码片段虽然不长，但写起来并不容易，需要花很多工夫，比如一个特定算法的实现、一种特殊的界面开发方式、一个手机上传大文件的代码片段。如果这些自己已经做过的工作不能在今后的项目中使用、借鉴，而要花工夫重新发明车轮是十分可惜的。

“看来曾经做过的还是收集起来比较好啊。”

“的确是这样，而且当我再看那些写过的代码时，往往仿佛无字天书一般。一个原因是时间长了确实忘掉了开发的细节，另一个原因是当时我几乎没写注释。”

“但至少是自己写的吧？会忘得那么多吗？”

“到时候你就知道了，所以说养成写注释的习惯对于开发人员是多么重要啊！”

作为一名合格的开发人员，追求问题解答的执着和热情固然值得肯定，但是总结错误、总结问题、积累成果的开发习惯也是必须要培养的。阔步前进的同时思索沉淀一下自己的思想，或许会让前面的路更加好走。

不写注释或注释过于简练可能是 IT 新人最容易犯的错误之一，写注释有时候就好比将自己高妙、复杂的思想翻译成通俗文字。由于很短的一段表述复杂思想的代码有时需要很长的注释才能解释清楚，所以很多时候爱炫耀的开发人员都不太愿意做这种工作，觉得太耗费时间。

连自己的思想有时候自己都会忘得一干二净，如何要求别人能看懂自己毫无注释的代码呢？如何在今后的开发中对项目进行维护和代码重用呢？

## 2.4.2 为自己维护一个小仓库

“蔡佳娃，所以说啦，以后在学习和编程当中，要不断地积累自己的成果，不要以为做过就算OK，将它们积累整理成一个小仓库，以后一定会大放光彩的。”

“嗯，我回去试着整理以前做过的一些小程序，我当时也是没有加注释，顺便验证一下是不是自己全忘了，嘿嘿。”

“呵呵，你会对自己的忘性叹为观止的，不过也不要什么都往里放，一些太简单的东西比如冒泡排序法的源代码就不必往里放了，有找到的工夫都可以自己重写了。”

“是啊是啊，不过我做的东西这么少，没有多少拿得出手的啊！”

“慢慢来嘛，只要是自己觉得比较有用的东西都可以放进去，有时候N年以后再看自己的仓库，也会不由地对当年菜鸟的自己有微微的嘲笑，呵呵。”

“希望我早点进步到那个程度啊。”

“肯定会有的，皇天不负苦心人嘛！”

很多开发人员在初期都没有很好地注意整理自己的小仓库，使得有一些做过的工作没有产生应有的价值，当需要的时候又要再起炉灶，重新开发。这样不但浪费了自己的劳动成果，降低了工作效率，有时甚至再也找不到当时的灵感了，十分可惜。

古语说得好：“不积跬步，无以至千里；不积小流，无以成江海。”各位有志于成为IT精英、高级开发人员的读者一定不要犯上述人群犯过的错误，平时注意多积累、多整理，当用到时就事半功倍，效率胜常人一筹了。

“嗯，要是早些听到你这席话就好了！我从今天开始就着手建立我的小仓库，看看多年后会不会被已成大器的自己嘲笑现下的愚笨，呵呵。”

“现在行动还不算晚，努力经营你的小仓库，小仓库也会时时回报你的。”

第二天，蔡佳娃拿着自己连夜搭建的小仓库给牛开复师兄检阅。

“师兄，你看我昨晚新搭建的小仓库怎么样？”（见图2-2）

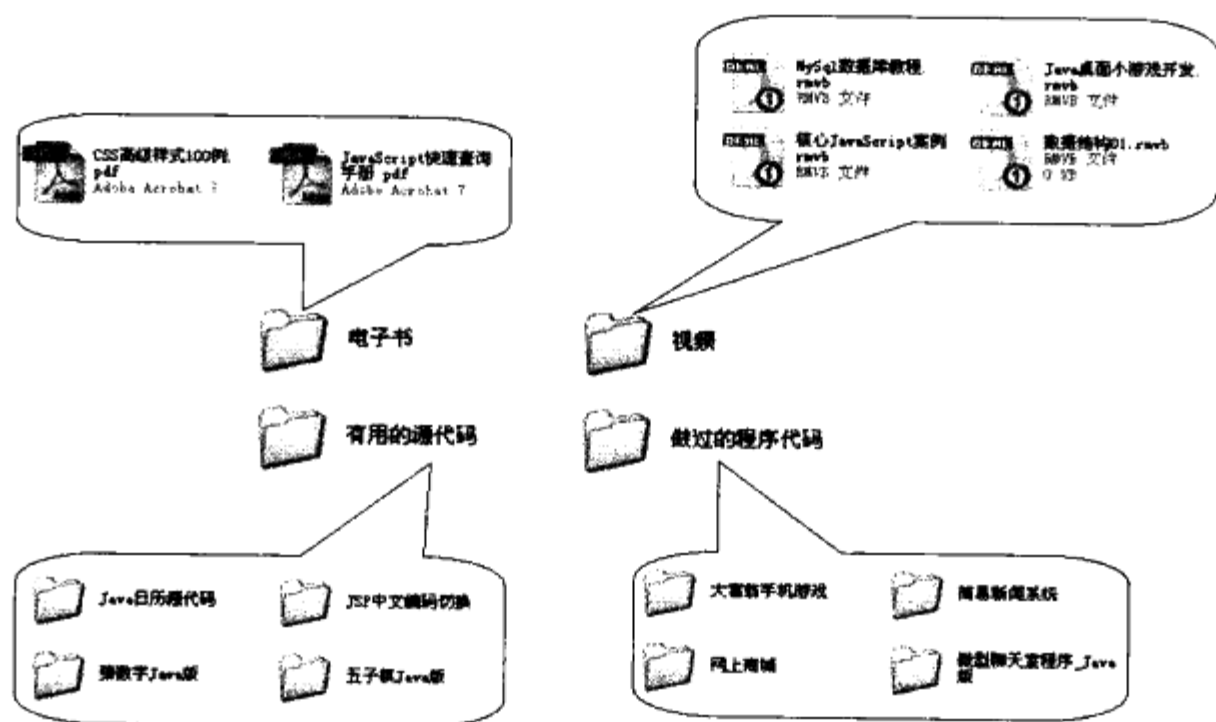


图 2-2 蔡佳娃小仓库图



“嗯，做得还算可以，不过内容不太丰富啊！”

“啊？这还不算丰富吗？”

“声明一下啊，我说的小仓库可不仅仅是电脑上的几个源代码的文件夹，这个仓库是可以有多种形式的，可以是书面文档，可以是电子书籍或者印刷书籍，也可以是各种技术文章、帖子，或是一段视频。多媒体时代嘛，你的仓库也要灵活丰富一些。”

“哦，原来可以这样丰富啊！可惜我之前对这些根本就不关心，只好以后上网或者读书的时候遇到好的东西时再留心收集了。”

“嗯，只要能找对方向猛冲，晚点出发也没有关系。这样吧，让你看看我的小仓库参考一下，这样你可以回去继续修葺一下你的小仓库。”（见图 2-3）

“嗯，师兄你对我太好了！呵呵。”

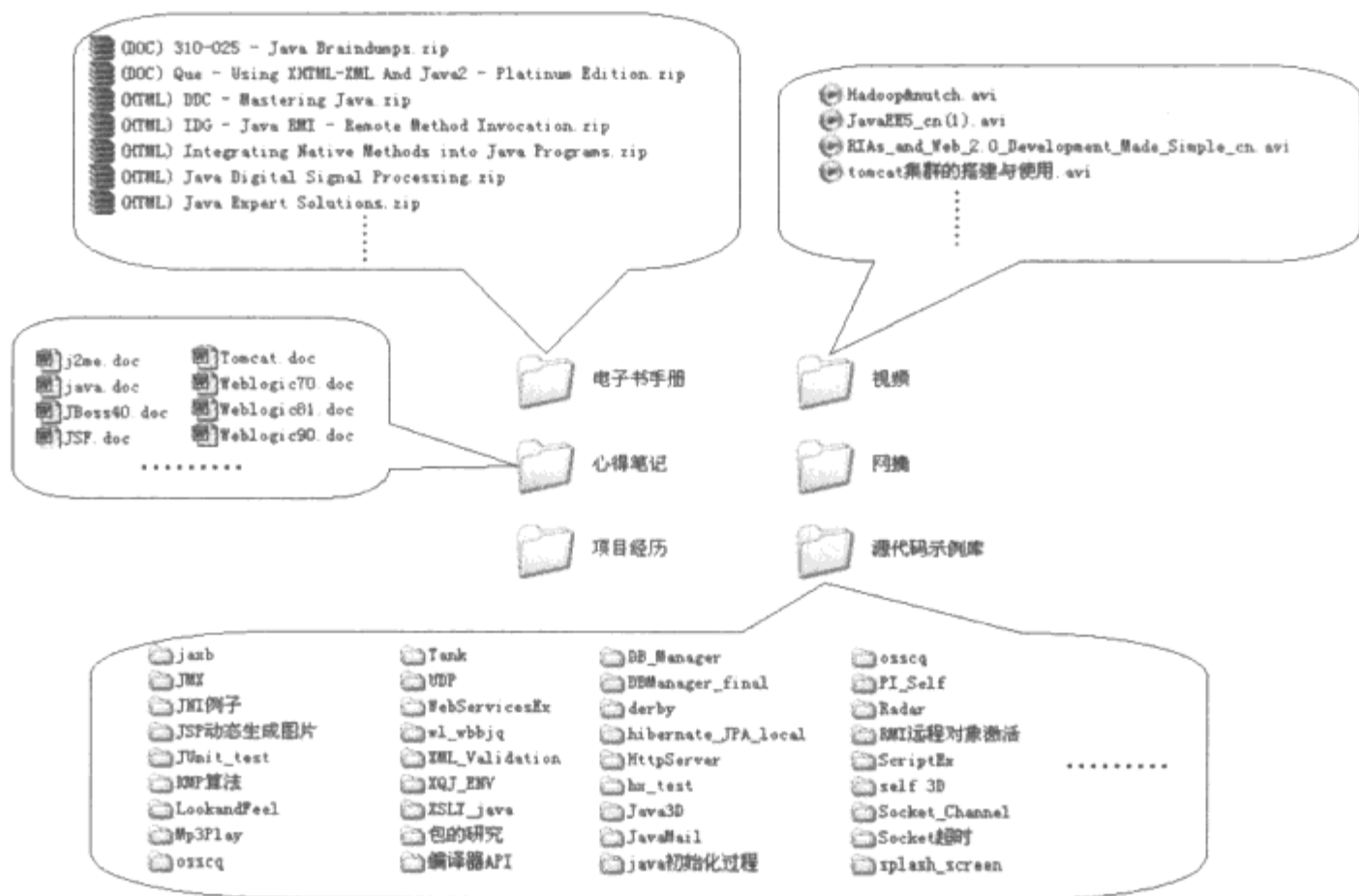


图 2-3 牛开复小仓库图

这个小仓库里装的，除了故事里说的那些自己做过的小项目、小程序的代码，还可以添加任何自己认为有用的代码、项目、文章、书籍等。

- 有些东西看了一遍，可能知识太深，自己能力还不到，不能理解透彻。可以将其存储起来等到自己能力到了，自然拨云见日，大叹“原来如此”。
- 有些东西自己压根就没时间看或是没有兴趣看，但是料到自己以后必定会用到的，比如看到一个 CSS 驱动的下拉菜单的代码，考虑到自己并不精通 CSS，但是经常进行 Java Web 的开发，可以直接存进小仓库而完全不用去细细研究，等到用的时候把案例调出来即可。
- 有些东西看了一遍，懂了，但是觉得很经典很高妙，对自己编程思想的提高有很大的帮助，绝对值得反复品味、感悟，这些东西也可以作为仓库的储备物。比如由国外大师 Erich Gamma、Richard Helm 等编写的《设计模式——可复用面向对象软件的基础》这本书，绝对值得每一位面向对象的开发人员时时反复品读。

### 2.4.3 多写开发心得

开发心得，或者称为编程笔记，对于很多新手来说可能很陌生，也很少有人会写。但是开发心得对于时刻检验自己、总结自己、完善自己有很大的帮助。同时，开发心得还是自己能力不断提高的见证。一个合格的开发人员会将自己的收获加到自己的开发心得中，作为一笔永久的财富。

“蔡佳娃，昨天回去搞了一晚上，你的小仓库修葺得怎么样了？”

“嘿嘿，还行，就是小点，我会慢慢丰富它的。”

“嗯，不错，世上无难事，只怕有心人。不过光有一个小仓库是不够的呀，你还要养成写开发心得的良好习惯。”

“我们在学校进行专业课实习和做实验都需要写心得，项目开发也要写心得吗？”

“首先，你必须与学校里胡乱凑字数的心得划清界限，此心得非彼心得。你也可以把它叫做编程笔记，总之就是一些总结性的东西。”

“那都是总结什么的啊？”

很多新手都觉得编程笔记很难写，不知道什么样的内容应该放进去。其实不用在这些次要问题上花很多工夫研究，只要自己觉得有用的都可以总结出来，写进去。编程笔记里面主要记录了你在编程过程中的收获和思想。

就像前面章节中讨论过的那样，这些内容时间一长会忘得一干二净，因此需要记录下来以备自己查阅。另外注意，编程笔记主要是给自己看的，就相当于一个备忘录，不必对格式、行文等次要方面有过多严格的要求，花费太多时间。

“那师兄，开发心得可以和别人分享吗？对别人有价值吗？”

“开发心得不仅对自己有用，拿出去对其他人一样也是很有实用价值的。我们公司的一个前辈，将近十年的 Java 开发笔记有二百多页，我曾经跟他开玩笑说，你稍微整理、润色一下，丰富一些东西，就直接可以拿去当书卖呢！结果他照做了，那书后来卖得还挺火呢。”

“哈哈，看来 IT 界最重要的还是技术思想和经验啊！”


“那是当然，好好做你的编程笔记，师兄我等着你签名售书的那一天！”

也许初期开发心得的技术含量并没有小仓库高，但是编程笔记里面记录的都是自己曾经有过的闪光思想或龃龉错误。或许小仓库的精华可以为你指明行进的正确方向，而编程笔记中的点滴却可以让你少走些弯路，加快速度，提高效率，甚至开辟新的康庄大道。

总的来说，编程笔记中可以写进去的有如下内容。

- 闪光的思想：不管是在漫长还是短暂的编程生涯中，总会有某一时刻的想法让自己眼前一亮，或许这种想法并没有解决当时的问题，但是谁敢说不会在自己今后的项目中发挥重要作用呢？到时候就可以暗自感叹：“原来我早就达到过这种高度了啊！”
- 错误和问题的记录：在开发过程中，肯定会遇到难以逾越的问题，或是容易犯的错误。问题解决了可以将其记录下来，等到再次遇到的时候就避免了绞尽脑汁也无法想起来的痛苦了；错误只有犯得多了才会记住，不过如果及时记录下来的话，可以让自己少犯几次。

- 经验的浪花：这或许是开发心得中最精华的部分了，编程知识如果是沙子，那么编程的经验无异于沙子中的黄金了。这里的经验并不是指开发过多少项目，而是在开发过程中通过实践得出的一些规则以及解决具体问题的方法。这些内容是哪一本教科书或任何一种课堂无法直接传授的。

 **提示** 笔者作为有多年开发经验的开发人员深深感受到，有很多经验在你没有接触过具体问题时就别人与你分享，你也很难理解。大部分情况下你还是会去犯那些错误，只有自己从实践中总结出来的经验才是最有价值的，读者朋友们平时一定要注意多总结。

## 2.5 本章小结

已经知道了暗礁在哪里，对于一个打算成为伟大航海家的人来说，所欠缺的或许就剩下勇敢地将船驶向大海了。或许了解暗礁在哪里，并没有学习如何开船来得爽快，但却会为航海增添一份强有力的保障。“天行健，君子以自强不息”，IT 这片神秘而又汹涌的大海，就等着你来遨游。

# 第 3 章 下山之路——有备无患

如果把 IT 这个新世界比作江湖的话，那么每个人在进入江湖之前，都必须在深山中苦练武艺。因为江湖险恶，稍有不慎就会迷失江湖。只有准备充分，才能让自己的下山之路更加平坦，让自己顺利地从小卒成长为江湖大侠。

## 3.1 从学生升级到开发人员

对于本书的大部分读者来说，进入江湖之前练功的深山，就是学校。如何从学校里舞刀弄枪的习武小子，成长为可以独当一面的江湖菜鸟，这个过程是每个人都必须经历的，也是很多人都颇为困惑的。本节将向读者介绍如何让自己从学生升级为合格的开发人员。

“师兄啊，当初你是怎么从整天上下课的学生直接过渡到每天上下班的开发人员的啊？”

“过渡？我没感觉啊？就是该干什么干什么呗！”

“可是我总觉得自己离一名开发人员还有很大的鸿沟呢？”

“鸿沟？没那么严重吧，看来我还得给我的小师弟做做岗前辅导了，呵呵。”

“其实就是觉得自己离着一个合格的开发人员还有些距离。师兄你指点指点我，让我有备无患吧！”

“嗯，我明白你的意思了。好，让我来指导你做下山之前的准备。”

### 3.1.1 学校给了你什么

学校，或许是每个人步入职场之前待得最久的地方了。不管学生对学校的态度是好是坏，是爱是恨，学校对于学生今后的发展都有很深的影响。但是，究竟这个占据我们生命至少五分之一的求学生涯给了我们什么呢？

“师兄，你看我都上大三了，怎么总觉得自己还是拿不出手呢？”

“很多时候是由于你对自己和 IT 职场不够了解的原因，不过我们之前已经探讨过很多这方面的问题了，从 IT 职场的揭秘，到 IT 职场的误区预报，你不应该这么迷茫啊？”

“这个我倒是懂，只是我不太明白大学这四年，包括之前上学这么多年学的东西够不够用啊？”

“原来你是对学校教你的东西不自信啊！看来还是让我这个师兄出马，助你提升一下功力吧！”

对于现在的大学教育，抛开那些只占全部大学一成的全国重点高等学府，可能所有人都会承认学校教育和社会需求出现了或多或少的脱节，不管他是学生、老师，还是校长。学校的知识或者是过多地落后于象牙塔之外，或者是理论讲了一大堆，一到实践就基本瘫痪。

但是脱节并不代表相去千里，不意味着毫不相关。纵观当今 IT 行业的人才需求，学校教育仍然是每个职场新手步入社会之前的必修课，这主要包含以下两个方面。

- 学校知识是基础

- 知识以外的收获

1. 学校知识是基础

“师兄，我觉得大学这四年在学校里待着，有些浪费。”

“嗯？为什么这么说呢？”

“学校里开的很多课都没有用嘛，什么高等数学、数值分析、大学物理、复变函数之类的，根本和IT这个行业无关嘛！”

“哎，我的小师弟啊，你如此不知书滋味，小心书到用时方恨少啊！在你眼中什么才是有用的呢？数据结构？网络原理？操作系统之类的？”

“不是吗？学就学点实在的，拿出来就能用上手的知识。”

“哈哈，我只能再次鄙视一下你的目光短浅了，不过也不能全怨你，现在的社会太急功近利了。你这种实用技术为上的态度我在大学里也有过，我可以告诉你我现在有多后悔当初没学好那些基础课，尤其是与数学相关的课程！”

不光是故事中的蔡佳娃，现实中很多大学生对于学校开设的基础课都不以为然，反感至深。比方说数学，计算机的指导思想和基础就是数学，早期的计算机科学家有很多都是数学家出身，而现在的高级计算机学者中也有很多人都在研究数学理论以提升算法效率。

需要说明的是，像数学、物理等基础课程中所讲的理论都是从长期的实践中经过总结提炼出来的，学习这些理论的目的就是在需要的时候将其再次应用到实践中。这就可以在很大程度上提高实践的效率，避免重新发明车轮，或者根本就发明不出车轮。

而IT又是个面向各个行业的服务性行业，客户提出的需求可不一定只限于IT这个专业，不是几个循环和分支判断就可以堆砌出所有实际应用的，因此能够掌握这些基础知识对于以后的快速、高效发展就大有裨益了。

“啊？原来那些东西还真的有用啊？”

“到时候你就知道了，我记得我在工作的时候不止一次回头翻看离散数学、概率统计的课本呢！那些你所说的没用的课，你是不是基本都没学好啊？”

“呃，也不是啦，只是兴趣不大，旨在不挂而已。其他还行，大学物理比较差，勉强及格。”

“看看，我记得你曾经说过对游戏开发挺感兴趣，那你知道游戏开发中渲染一个3D场景需要做什么工作吗？”

“不是直接写代码吗？”

“很多地方都要解薛定谔方程！估计你连这个名词都不记得了吧！”

学校里每一门学科其实都是面向应用的，只是在教学的过程中或者是课程的安排有问题，或者是老师能力的缘故，只注重了知识的传授，忽略了将其融会贯通，结合起来联系实践。因此有些课程看起来似乎太离谱，几百年也不会再用到。

很多人在这种情况下表现出来的逆来顺受会让人很难理解，而有些人也会和故事里的蔡佳娃想的那样，既然课程不好，或者老师不好，干嘛费心学呢？不如学点自己认为有用的东西。但这样做的结果最有可能是丢了西瓜捡芝麻，捡来后却发现还是个变质的芝麻。



“师兄你说得对。那些课程我是没有好好听，当时我想更好地利用时间，所以都会自己去看一些有用的书。那段时间我从图书馆借了好多书呢。”

“好，那我问你，借了那么多本书，从头到尾看完的有吗？有几本？”

“这个嘛，呃，好像没有吧，都是看一大半就被什么事情打断了，或者看到后面内容太深看不懂，或者是该考试了，得抓紧复习了。”

“所以说嘛，有的时候为了抓紧时间双管齐下，认真学习课本知识的同时又大力钻研自己认为更有用的技术，但由于计划混乱，最后的结果往往是兵败两地。你应该好好利用在校学习的时间，每一门课都或多或少有它存在的价值，你可以抱怨学校对你不负责任，老师对你不负责任，但是不可以自己不对自己负责任啊！”

“嗯，师兄你说得太对了。我以后得好好规划一下自己的时间，让自己不再竹篮打水一场空。”

“对喽！学校的知识不是万能的，但不会是万万不能的呀！”

在学校所学的都是基础，虽然效果不像学一门实实在在的技术那样立竿见影，但是学校的知识会在潜移默化中提高一个人的思维能力和理论功底。比方说数学，数学是解决问题的一种抽象思维，现实中很多问题都可以理解为数学模型来解决。所以说数学是思维的体操，一点没错。

例如游戏中要开发水波纹的场景，就要用到与波方程相关的数学物理知识；应用中要优化一组解，就要用到最小二乘法等。如果在学校学习时不重视，等到要用时不但不会，甚至都忘记它们的存在了，那就很难胜任工作了。

在这里提醒那些完全抛弃学校教育而只靠自学的同学，自学精神的确可嘉，也很有必要，但是绝不可以彻底否定学校的教育。不是学校给你的知识不好，而是你没有用心去学。或许课本知识没有学习一门技术有激情，但是技多不压身，可以不精通，但是不能听都没听过，没准哪天会因为你比其他人更了解薛定谔方程或自动机原理从而会脱颖而出。

## 2. 知识以外的收获

“师兄，听你这么一说，我今后还是踏踏实实地学些课堂知识，要不然内功修为不够啊！”

“这样很好，不过说实话，学校的课程里面还真是有一些枯燥无用的，或者是彻底过时的，这些课程你自然可以抱着六十分万岁的心态来对待咯。”

“是是，我还是能分清什么课程是可以放弃的。”

“那样就好，其实学校除了给予你出来混所必要的一些基础知识以外，更重要的是养成固定的、良好的习惯。”

“习惯！这个和以后的工作有关系吗？”

“当然有了，我所指的习惯包括生活、学习、思考、做人等各方面，大学生活中养成的习惯很多时候会跟随你一辈子呢。”

“啊，有那么大的影响吗？”

大学是走向社会的最后一个准备阶段，在大学的四年学习生活中，每个人都会在性格爱好、学习思考方式、待人接物甚至生物钟方面形成一些固定化的模式，这些就是习惯。习惯一般是最不容易被注意到的，很多时候习惯一旦养成就很难改变。

习惯成自然，或许这句老话已经过时很多年，但是道理却一直都是正确的。很多职场新手有

时不是败在自己技术不行、脑子迟钝、经验不足上，而是其或大或小的一两个沿袭自学校的坏习惯使自己自断神兵。

“我给你举个例子吧，我们公司前些日子新进来一个大学毕业生，技术能力不错，脑子也够灵活，可办事总是拖拉，主管分给的任务他总是最后一个完成。”

“哦，那是为什么啊？”

“主管也很纳闷，本来很聪明能干的人，不应该表现得这么拖拉啊。于是主管叫我观察一下这个新员工，结果我发现了他的秘密。”

“什么秘密啊？”

“这个人的确非常有能力，我见过他写的代码，很规范很华丽。可是我发现他无法让自己的注意力集中时间超过30分钟，30分钟后，他就有些开小差，查阅一下邮件，接杯水，浏览一下其他网页等，就这样杂乱无章地玩一会，然后再继续工作。”

“所以他把时间都给浪费了？”

“是啊，然后我就问他为什么会这样。他说他在大学的时候就是这样，刚刚开始学编程的时候，每次写完一段难度很高的代码，就很有成就感，就想奖励一下自己。而且他宿舍的其他人一般都在玩游戏，他也受一些影响。”

“啊，看来学校里面养成的习惯真会带到工作中啊！”

“是啊，后来这个同事花了很长时间才跟上我们的工作节奏，要不是因为他有点天分，估计早把他辞掉了。”

除了故事中提到的那个无法长时间集中注意力的坏习惯之外，还有一些坏习惯需要避免。比如一看书就头疼打瞌睡、大早起来特别困等。都说21天可以养成一个习惯，然后用90天的时间使之趋于稳定，不知道改正一个坏习惯是怎样的流程，估计比养成习惯难N倍。

同时，在学校的学习中，也会逐步建立起比较专业的思考问题方式，并逐渐将自己的思维模式固定下来。经过大学的历练，不仅知识储备上必须达到要求，个人的学习工作习惯和思维方式也应该提升到能胜任合格的开发人员的水平。

“师兄，听你这么一说，看来学校对我们的知识和思维能力的潜在影响还真是不小啊！”

“是啊，另外，学校虽然是个教书育人的地方，但是现在的大学也越来越像个小社会，在大学的几年时光，或多或少地也教会了我们一些为人处世的道理。”

“嗯，师兄你说得没错。有时候如何处理好跟其他人的关系，也是比较考验人的。”

“那是当然，其实到了社会上也大概就是这个样，同事就像同学，既要团结友爱，又要互相竞争；老师就像上级，既有敬佩听话，又有怀疑不服。虽然学校远没有社会上那么严重，但是至少在学校能让你提前感受一下这个气氛。”

“是啊，师兄，说得很有道理。做人难啊，呵呵。”

“做人才更难，做个好的IT人才，那是难上加难啊，哈哈。当然了，学校也没有如此地不近人情，它主要还是提升我们各方面素质的地方。”

从学校获得知识的同时，也收获了很多习惯，养成了一些基本固定的思维方式和做事的准则。学校虽然不能为你以后的工作做百分之百的保证，但是如果你没有荒废学校的教育，那么就算万

丈高楼倒下了，至少地基还是在的。

总结一下，从学校中应该获得的有以下几点：

- 基础知识，包括专业知识和其他领域的知识；
- 学习和工作的习惯；
- 思考问题及做事的方式；
- 为人处世的一些学问。

### 3.1.2 咱们还缺啥

学校教育是走向职场的基础，这一点是可以肯定的，但是只有学校的教育够不够，这就见仁见智了。IT 行业是个技术发展、更新很快的行业，尤其是软件开发方面，新的技术层出不穷。因此，学校这个比较庞大的列车在处理这些新技术时偶尔就会显得有些力不从心了。

“师兄，上次你跟我说不要小看了学校教育，要让自己的基础知识尽量扎实，但是其实光靠学校教育是远远不够的吧？”

“就我们 IT 行业来说，的确直接从学校教育走出来的学生离企业需要的人才还是有一些差距的，脱节现象比较严重。企业需要的大都是不管材质如何拿来就能用的成品，而学校培养出来的大部分都是材质不错但是欠缺雕琢的半成品。”

“所以在学校知识之外，我们还得掌握一些必备的专业知识来应付求职之路。”

实际上，如果学校开设的课程还算合理有用，不是完全的落俗套、做表面文章，那么把学校的知识彻底掌握，是完全可以找到一份比较满意的工作的。因为具有扎实的基础知识的人才相比较于那些故事中提到的只注重表面技术的技巧型人才，同样也是有一定人气的。

但是在一般情况下，从学校顺利毕业后还是很难按照自己的理想满意就业的。这里面的原因很多，学校、社会有责任，但更多还是在个人。这也是为什么如今的 IT 培训和认证考试如此流行和受欢迎的部分原因，认证将会在后面为读者介绍。

“正解，但从掌握知识这方面来说，大学生毕业时应该至少精通一门编程语言，会用其编写简单应用，熟悉数据结构和算法方面的知识，以及计算机网络和数据库方面的知识。”

“光看那些程度词‘精通’、‘熟悉’就知道企业对于人才的要求还是蛮高的啊。”

“对啊，所以不能再像以前应付学校那样来应付公司啦。我刚说的这些知识，或者是学校里没有开设相关课程，更多的是有相关课程但是学了不够用，或不会用。”

“师兄，你说得太正确了！我对这个深有感触，曾经学了 C 语言后觉得自己很牛，然后去网上的论坛里瞎逛，发现人家讨论的问题我压根都没听过。”

“是啊，种种原因吧，学校没有办法让我们进行最充分的岗前训练。我们在踏入职场之前最好是在学校知识的基础上深化一下自己的知识，这样才能顺利搭上 IT 的特快列车。”

学校教育是面向大众的，因此要兼顾不同水平的学生，所开的课程就不可能过于高深。况且学校的教学水平也是有高有低，对于新技术的反应也有所不同。所以很多时候学校教育和企业需求之间的鸿沟需要求职者自己来填平。

企业对人才所具备的知识需求主要有以下几个方面：

- 编程语言方面，需要熟练使用一些辅助语言，至少精通一门主编程语言。这里的辅助语言主要指那些通用性高被广泛应用的语言，如 SQL、HTML、CSS 等，这些语言一般功能比较单一（如 SQL 只面向数据库处理），但是却具有不可替代性，所以必须熟练使用。而需要精通的主编程语言则是那些功能强大的高级语言，如 Java、C/C++ 等，这些语言将会成为进入职场后开发时的主要语言，所以必须毫不含糊地彻底拿下。
- 数据结构和算法以及设计模式，这些是永远都不会过时的内容。武侠小说中经常会提到的“万剑归宗”的“宗”，在计算机技术中应该就是算法和设计模式了。编程语言只是一种工具，而算法和设计模式则是语言无关的。
- 计算机网络原理和通信协议，随着网络技术的发展，不具有网络连接功能的软件已经基本退出了历史舞台。因此要想做一名合格的开发人员，网络原理的知识必须了解，而通信协议的内容则必须熟悉，并能够根据其原理设计开发自己的网络应用协议。

“师兄，听你这么一说，我们离公司招聘的需求还是差得不少呢。”

“呵呵，所以说剩下的这两年还是不能放松啊！”

“那是必须的啊，不过师兄，提升自己、弥补学校教育缺憾的途径是什么呢？”

“一般来说，有三种方法：自学、网上获取资料、参加培训。首先说说自学，自学算是成本最低的一种提升方式，但也是一种水滴石穿的过程，因为自学全靠自己的领悟，而且自学需要对自己的约束能力很强。自学能力也是一种很重要的能力。”

其实一个人的自学能力一般也是从大学开始培养起来的，进入大学后学生需要自己主动地学习，家长和老师一般不会太多的干预。而进入职场后无论是忙中偷闲努力缩小和别人的差距，还是快马加鞭大胆尝试新技术，靠得都是自学能力。

而且最新的技术一般是没有什么完善的教学体系和资料的，也很难从别人那里学习，能否学好用好新技术主要靠的就是自学能力，因此读者朋友们一定要注意自学能力的培养。

“说得没错啊师兄，我的自学能力就不够好唉。”

“很多时候自学能力不强是因为压力不够大，你慢慢就知道抓紧了。一般情况下自学都和第二种方式联系起来，就是网上获取资料。”

“嗯，我想也是，现在网络上什么都有，肯定特方便。”

“对啊，网上获取资料包括在社区论坛里面提问讨论、从网上获取各种形式的电子教程和源代码、了解相关技术的最新消息、获取各种技术支持如开发环境和源代码等。”

“是啊，我的电脑里还有很多视频教程呢。”

“网络上的资源相当丰富，而且新技术的介绍也是网上快于书面，所以不利用这个资源肯定是个浪费。不过网上资源和自学有个比较大的缺陷，就是交互能力不够强。”

“交互能力？”

“对，就是在学习的过程中遇到难以逾越的问题，在学校教育的模式下会去向老师询问，就是和高手交互，而自学和网上获取资源由于没有老师，所以这方面显得有些不足。”

“可是，可以在网上提问题啊！现在这种网站很多啊！”

现在这种问答形式互联网服务很流行，而且查询起来也非常快捷，但是有些时候，在网上提

的问题很少会有让人满意的回答。有时获得了满意回答后，想进一步讨论也很难。

最一般的情况是，很多人的回答要么简练无比不知所云，要么答非所问一头雾水，偶尔有一两个回答很对路，进一步追问追问，结果人家再也没回过帖。不可否认的是，网络上不乏高手，但是能找到一个真正能够排忧解难的人恐怕很难。

而且要注意的是，网上能有人给出答案的问题一般都是对菜鸟特别有用。比如帮助修改几行源代码、重新配置服务器等，这些对成手而言都不是问题。真正想要在网上根据自己的需要讨论复杂一些的思想性问题是很难的，这点笔者深有体会。

“师兄，听你这么一说，想想我曾经在网上苦等一个满意答案的悲惨经历，网上问题解答的方式还真不是非常有效的呢。”

“从网上获取一些小问题的解答是很方便的，比如 Tomcat 服务器的配置等，所以还是要对网上的资源敞开怀抱的。最后再说说培训这个途径。”

“这个我听说过，培训就是花钱去学技术。”

“是这么回事，不过培训相比前两个途径的优势就是交互能力强，因为培训一般是传统的老师和学生面授的形式，所以在解答问题等方面是很有优势的。”

“是啊，比网上更快，呵呵。”

“同时培训比较符合学生的学习习惯，毕竟人们在网上学习才几年，而在教室里上课的历史可是很长了。再一个原因就是参加培训一般要花上一笔可观的学费，这也应对了我刚才说的有压力自学能力才会更强，因此参加培训的学员会比其他人自学起来更有动力。”

在真正的学习中，最好还是把三者或者至少前两者有效地结合起来，尽量提高自己的水平，使自己在学校知识的基础之上补漏拾遗，更上一层楼。

### 3.1.3 经验，还是经验

搞定了学校的基础知识，又提升了自己的能力以达到标准，最后来讲一讲让很多即将毕业的大学生谈之饮恨话题：项目经验。很多公司在面试的时候基本上都会问到的问题就是：谈谈你曾经做过的一个项目。所以为了让自己成功跨入开发人员的行列，经验还是必须要有的。

“师兄，经过了学校的学习和自己的努力，应该可以满足一个招聘单位对求职者的要求了吧？”

“嗯，知识上是差不多了，实践方面还是需要加强，比如要有一定的项目经验。”

“啊？最后还是要求经验啊？我们刚刚毕业，哪来的经验啊！”

“这也是用人单位没办法的事，用人单位其实只是想要一个技术好、有能力的人，可是技术和能力这东西短期内也不大容易看得出来。所以加个项目经验的条件，自然会让其招进来的员工更加保险一些。”

“是啊，现在职场竞争这么大，无经验无门路啊！做大学生难啊！”

现在可不是抚琴慨叹、怨天尤人的时候。说得残酷些，那些零经验进入职场的人，在学校考零分和满分是没有差异的，因为他们都将面临淘汰的可能，就好比去开发房地产，地皮买好了，打个地基，插几个桩子就开始卖房收全款，那样不现实的做法肯定不会有人买账。

因此，在校大学生朋友们虽然在踏出校门前很难有工作经验，但是通过自己的努力获取一些



项目经验还是十分有必要的。获得经验的大致途径后面会介绍，这里读者不必担忧。

“是啊，这个说得倒是没错，我之前照着一些书上的内容自己写程序，书上的都看明白了，自己真正写的时候就会蹦出这样那样的错误，有时候跟书上的东西一模一样，但是到最后还是会发现忽略掉的细节和错误。”

“‘纸上得来终觉浅，绝知此事要躬行’，IT 行业对实践能力的要求是相当高啊，所以说为了自己保险起见，你也要有一些项目经验才行。”

有句古话说得好，“腹有诗书气自华”，真正有能力的人无论走到哪里，都像黑夜中的萤火虫一样明亮显眼。但是还有一句古话：“千里马常有，而伯乐不常有”。虽然你很闪亮，但是有可能没有碰到赏识你的人，所以项目经验这个实实在在的东西就显得非常有必要了。

这里有必要区别一下知识与技能之间的不同，因为掌握了知识并不代表具有了技能。知识和技能的转换关系可以用如下的方程式来表示，如图 3-1 所示。



图 3-1 知识与技能转化图

图 3-1 的含义就是知识在反复应用的条件下会生成技能和经验，很多求职者都停留在方程式的左边，而用人单位往往需要的是右边的技能。知识在转化为技能的同时也会丰富一个人的经验，而相对于技能而言，一个人的经验如何往往更容易检测出来。因此很多情况下，根据求职者的经验介绍，用人单位就可以对其技术能力的水平做一个大致的判断。

“师兄，那我们这些人应该怎么混出一些经验呢？”

“每个人提高自己水平、积累经验的方式都不一样，我来讲讲当年的经历吧！”

“嗯，我们这种菜鸟最喜欢的就是举例子。”

“我在大三下半学期的时候，觉得自己知识储备得差不多了。当时没有人可以投奔，也没有人给我任务，所以我就花时间给自己出了个题目让自己做。”

“那是什么题目啊？”

“一个图书馆的信息管理系统，做完那个模拟项目的确让自己进步了很多。后来学校里一位老师接到了一个项目，需要帮手，接着我就去了。老师因为看到我之前做的那个图书馆信息管理系统的的项目还可以，就留下我了。”

“这算是师兄你最初的项目经验吧？”

“差不多吧，老师分了一些任务给我做，之后我又帮着那位老师做了几个项目。在大四下学期开始的时候，我已经在一家老师推荐的公司断断续续实习了将近一个学期了。”

“是啊，这么一来师兄你的境界和那些一直闷在学校里的同学可就大不一样了啊。”

“呵呵，这些你可以听听，具体怎样让自己的知识转化为技能和经验，还是要靠自己的努力呀，记住你曾经说的话：无经验无门路。”

“嗯，师兄，我会找到自己的提升道路的。”

从学生到开发人员的升级之路，说起来很简单，但是每一步做起来都不容易，找工作最好的境界就是胸有成竹，其实成竹在胸的就是技艺精湛、经验十足的底气。总结一下这条下山之路首

先要完成的任务有如下三个：

- 学校获得的知识或其他东西是自己进入职场的“地基”，没有地基的房子固然不一定会立即倒塌，但必定是鲜有人问津。
- 不断提高自己以满足不断变化的企业需求是进入职场的“发动机”，说到底主动权还是在自己，能不能让自己发动起来就要看自己的干劲了。
- 多写多练注重实践经验的积累是进入职场的“助推器”，既然是助推器，貌似没有发动机重要，但是却决定着开发人员这辆车的速度与能效。

## 3.2 为自己定下目标

对于每个将要踏入 IT 这块战场的菜鸟来说，为自己定下一个目标并努力去实现将会是一个非常好的进步方式。如果把一个人的职业生涯比作一个巨大项目的话，制定不同的目标就是在完成不同的模块，只是这个项目只有自己才能完成，而且没有别人设定的进度限制。

### 3.2.1 目标的意义

“蔡佳娃，今天我们来谈一个比较崇高的话题。”

“什么崇高的话题啊？”

“目标，在进入职场前要为自己定下目标，并且努力去实现它。”

“哦，这话题也不算崇高嘛。”

“目标是一种务实的说法，我如果说理想是不是就有崇高的感觉了？呵呵，闲话少叙，你有没有为自己立下过什么目标啊？”

“这个怎么说呢，目标倒是定过，不过我没觉出来定不定下目标有什么区别啊？”

“当然有区别了！哦，我明白了，看来和你讨论目标之前还得先奠定一个基础。”

“啊？什么基础啊？”

“就是目标是用来干什么的啊？”

如果给别人解释目标的含义肯定很少有人喜欢听，因为目标的含义很简单，就是一个人行动的方向和期望达到的结果。目标有长有短，有大有小，我们时时刻刻都在为着某个目标努力，长到一个终身奋斗的理想，短到一两个小时的工作计划；大到引领一番变革的风浪，小到给自己一个满足的欣慰。

作为一个即将投入 IT 浪涛中的新手，最好为自己定下一个目标，而且必须是个不能太小且长期的目标，这样才能找到前进的方向，不至于在狂风颠簸中迷失自己。

“所以啊，蔡佳娃，你要为自己定下一个长远的目标，这样你用起功才会无怨无悔，甘之若饴啊！否则每天只停留在自己的小计划中，慢慢也就会失去工作的激情。”

“嗯，师兄说得对，没有目标就跟没有舵的船一样，迟早会迷失的。”

“正解，纵观历史中风起云涌的各路英雄，偶然成功的人有，但更多的还是定下长远的目标并一步一步按着自己的雄心走出来的。”

“是啊，就像武侠小说中写的一样，世外高人、怪侠异人层出不穷，但是统领江湖、长久不衰

的，还是那些由小及大、逐步发展壮大名门正派的。”

“呵呵，你武侠小说看多了吧。下面我给你讲一个靠远大目标赢得长久胜利的英雄故事，故事的主人公叫做帕特里克·麦戈文。”

帕特里克·麦戈文这个名字或许不像比尔·盖茨和史蒂夫·乔布斯那样鼎鼎大名、如雷贯耳，但是或许大家读过一本杂志叫《计算机世界》，《计算机世界》的创办者就是麦戈文。正像比尔·盖茨打造了微软帝国一样，麦戈文作为IT界的传媒大亨，打造了IDG帝国。

如果读者有兴趣，也可以到图书馆找几本《计算机世界》来读一读。读过就会发现，《计算机世界》还真是IT界很有影响力的杂志，从杂志中可以了解很多行业的最新动向与发展趋势，这对我们IT人来说是很有帮助的。

“麦戈文首先是一名资深的IT媒体记者，与其他IT媒体经营理念不同的是，麦戈文创办IDG（国际数据集团）的时候，就将公司的发展目标定为面向全球的具有国际性质的IT信息提供商，这也是其主打刊物《计算机世界》得名的部分原因。”

“树立了这样的目标，麦戈文便开始了自己创办世界级的IT舆论阵地的王朝之路，当时的IT杂志都是靠广告费生存，谁给的钱多就为谁说好话。麦戈文看到了这种商业模式的弊端，在1967年创办了《计算机世界》杂志，由此标志着这个帝国的崛起。”

“凭借着他对市场的敏锐嗅觉和资深的IT媒体记者经历，《计算机世界》迅速成为全美最受欢迎的IT杂志，在公司成长到一定规模之后，功成名就的麦戈文马不停蹄地继续实现着他的宏伟目标，在1974年将《计算机世界》带进了欧洲。”

“随着IDG公司在IT媒体界的舆论影响力与日俱增，麦戈文将自己的出版物发行到了全球100多个国家，推出了20多种语言版本，分公司遍布世界各地。”

与此同时，很多当时和麦戈文的《计算机世界》同时起家或者更早的IT媒体，由于没有像麦戈文那样早早地定下全球战略，都在IDG的巨大光环下或慢慢退出，或被收购。而麦戈文一手创建的IDG由于正确的目标指引，成为当今IT世界最有影响力的媒体平台。

“蔡佳娃，听了这个故事，你是不是有点想法了？”

“嗯，师兄，虽然我做不到麦戈文那样的IT媒体皇帝，但是我也会为自己立下一个目标，努力前进，或许一个目标一个目标的实现，有一天我也会发现，我成功了。”

“嗯，你有这样的想法非常好，师兄期待着看到你的进步！”

目标促使一个人将准备要做的事情细细规划并实现，一个人只有明确了目标才能找到前进的动力，才能为自己付出的异于常人的努力找一个合理的借口，才能让自己在别人的不理解和迎面而来的困难面前不为所动，持续前进。

### 3.2.2 树立目标的学问

前一小节提到了目标的重要性，但是很多时候导致失败的并不是没有目标，而是目标不够合理，或者目标太多，又或者目标频繁变动。因此，如何立下正确的目标也是个马虎不得的重要过程。

“师兄，上次你只是说了定目标的重要性，那么定目标时需要注意的问题是什么啊？”

“首先，目标要恰当，要适合自己，不可盲目自大，也不可妄自菲薄。要首先对自己有正确的认识，才可以立下合适的目标。”

“嗯，我估计我有点不太自信，可能立的目标会比较屈才。”

“目标定得过小，你的才能就无法彻底发挥；而目标定得过于飘渺，有可能会让自己的前进道路荆棘遍地而困难重重，最终无法实现。”

“所以要尽量避免做一个眼高手低的人是吧？”

“正解，有些人的目标过于宏伟，结果在向着目标前进的过程中总是碰壁，碰来碰去，不仅鼻子扁了，意志也被消磨掉了，离成功也就会渐行渐远。”

目标定小了，实现起来会容易一些，也可能会增添一些信心，但是也有可能变得夜郎自大，认为所谓 IT 精英也“不过如此”；而目标定大了，就有些不切实际，往往会在不可避免的挫折中丧失信心，以失败收尾。

在实现目标的过程中，发现错了，不要对自己太容易懈怠和放弃，也不能硬着头皮死扛，而是要立刻进行修正，以免自己的努力付诸东流。

有些人在立下目标的时候只是逞一时之快，感受一下自己的豪情万丈和一腔热血就了事，之后便又回到原来的老样子，等到哪天又受了刺激再来一次振臂高呼，好像目标只是一首励志歌曲，可以唱在嘴里，却无法做出来。

“另外给自己定目标时还要注意的是不要频繁地变动当前的工作中心，爱迪生不是说过嘛，‘比起有一千个有创意的想法而一个都不做，我更喜欢只有一个创意而努力去实现的人。’所以说目标不管大小，不随意改变也是很重要的，贵在坚持嘛！”

“是啊，我们的导师也经常说不管大家选择考研还是就业，一旦选择了，不要变就行。”

“说得没错，有些人就喜欢变来变去，今天看了李开复的演讲视频就拍案而起满腔豪情地立志要做个有为青年。刚刚发愤图强了几天，四六级考试又迫在眉睫了，只好先救近火，全身心地背单词，考过四六级之后又会有新的事情加塞进来，如此反复……”

“师兄你分析得很透彻啊！”

“呵呵，下面我再给你讲个 IT 界伟大思想家的故事吧。他的名字叫做道格·恩格尔巴特，不知道你听过没？”

“没有唉，我这个人太孤陋寡闻了。”

其实也不能怪蔡佳娃孤陋寡闻，道格·恩格尔巴特或许是 IT 历史上最容易被忽视的一位思想家和发明家了。恩格尔巴特在几十年前提出的非常有预见性的技术和观点都不被当时的主流计算机学者所接受，而现在计算机能有如此友好的人机交互界面全都是拜他的坚韧努力所赐。

关于鼠标的发明者，很多人只知道是一个海军军官，其他的便一无所知。其实恩格尔巴特就是鼠标的发明者。当我们今天在电脑前自如地使用鼠标的时候，却不知道它的发明者为此经历了多少的挫败和打击。而如今他对于计算机发展的设想都广泛地应用在每个人的生活中，而不仅仅是鼠标。

“恩格尔巴特接触计算机技术是在他 25 岁那年。之前他一直在美国海军服役做一名雷达兵，退役后他做了一名电气工程师。当时计算机的输入输出靠的全是穿孔卡片，而天才的恩格尔巴特

却开始有了为其配置输出屏幕等设备以增强人机交互能力的超前想法。”

“恩格尔巴特为了他的目标开始了奋斗，他先在加州大学伯克利分校攻读博士学位，而后又在斯坦福研究所工作，还到社会上去寻找能支持他的合作伙伴。但是所有他遇到的人都认为他的想法不可理喻，也没有几个人能听懂他的理论。”

“那不跟爱因斯坦当年提出相对论一样吗？”

“的确是这样，与爱因斯坦一样，恩格尔巴特依然没有放弃自己的梦想，终于美国军方开始对他人机交互的研究感兴趣，并给了他一笔经费做研究。在进行研究的初期，恩格尔巴特也是孤军奋战，直到后来他的文章逐渐引起人们的注意，不断有人为他的研究提供资金，他才开始真正地铸造自己的梦想。”

“1968年恩格尔巴特终于等来了机会，在一次电脑会议上，他向观众展示了他的研究成果，其中就包括鼠标的亮相。这次会议绝对值得被载入IT的史册，因为恩格尔巴特在那次会议上演示的人机交互思想与技术，在从那之后的几十年里都如火如荼地发展起来了。”

“或许是天妒英才，正当恩格尔巴特的研究处在重要阶段的时候，他的研究中心失去了资助，很多人都离开了，最后又是剩下固执的恩格尔巴特继续为了自己已经奋斗几十年的梦想而努力。”

到了20世纪末，恩格尔巴特设计的具有友好用户界面、多窗口显示器、鼠标等人机交互能力的计算机系统终于获得人们的认可并迅速发展起来。这时，人们才意识到原来几十年前这位杰出的思想家和实干家就已经在默默地为之奋斗。

尽管如此，恩格尔巴特并没有获得与他的贡献相当的名誉和财富，他仍然继续追求着很早之前就为自己立下的目标，或许这样他才能更加快乐。

有句古语说得好：“小人恒立志，君子立恒志”。总结一下，在为自己定下目标的时候遵循的几个基本原则如下：

- 量身打造，适合自己。
- 不能白立，切实真干。
- 不在宏伟，贵在不变。

### 3.2.3 让自己知道今天该干什么

如果将每天的成果比喻成砖瓦的话，有目标和无目标的奋斗就犹如长城和砖垛一样。心中有目标，每一步的实现必然是向着成功越来越远，最终将一块块砖瓦垒成了万里长城；而没有目标的努力，虽然也会生产出漂亮的砖瓦，但却仅仅是整齐地摆放在一起，永远都还是只是原材料。

“师兄，即便是定下了目标，做了长远的规划，我还是对于每一天该干的事情很茫然。光看着目标在前，但却追不上去。”

“有你这种想法的人也不在少数。目标有了，但具体到当前这一天、当前这一小时该干什么，就有些犯难了。”

“那师兄你帮我解答一下呗。”

“别急，我还是再来给你讲个故事吧！”

没有目标的时候迷茫，有了目标后却彷徨。很多立下目标的人可能都会有这种感受，苦于不



知道该如何继续自己的奋斗。读完下面这则类似喜剧的小故事，或许能给像蔡佳娃这样茫然不知所措的人一些指导性的意见。

一位女士（姑且叫做 Mary）刚刚过完自己的 30 岁生日，但她并不开心，于是，她找到了自己的好友吐露自己的烦恼。Mary 告诉自己的好友，现在的状况完全不是自己很早之前规划好的。好友就问她原先的规划是什么，Mary 说她的想法是在 32 岁或 33 岁的时候已经做妈妈了，但是她现在非但没有结婚，而且正在交往的男朋友比她年龄还要小，短期内根本没有结婚的打算。

好友说：“那你是不是很想坚持你自己的计划呢？”

Mary 说：“是的，我就是为这个而沮丧的。”

好友说：“那我们来分析一下。你想想看，我们假设你要在 33 岁的时候做妈妈，那么你应该至少在多少岁的时候结婚呢？”

“至少 32 岁，最好 31 岁。” Mary 回答说。

好友接着问：“那我们假设你在 32 岁结婚，那你应该在多少岁的时候认识你现在的丈夫呢？”

“至少两年前，也就是 30 岁的时候。” Mary 说。

“那你现在遇到了没有啊？”好友继续问道。

Mary 想了想自己现在的男友，无奈地摇摇头。

好友说道：“分析到这，你应该明白现在该干什么了吧？”

Mary 想了一会，眼前一亮说道：“嗯，我现在应该和现在的男友分手，努力让自己去结识一个打算结婚的男士去！”

故事中 Mary 女士的目标就是最晚 33 岁的时候当上母亲，而 Mary 所做的却不是朝着目标在前进，甚至是在慢慢偏离自己的目标。Mary 的朋友帮助她巧妙地分析了一番，在时间上从将来往现在推导，由远及近回到现在，Mary 才发现自己当下应该处于什么阶段。

故事虽然有些搞怪，但是其中的道理却值得提炼出来深思。不像一般人从现在开始幻想到将来，我们可以从将来的某一天往现在回溯，这样自己职业生涯中每个阶段的状态都清晰明朗起来。将长远目标彻底分解成一个个的阶段状态，这样就不会再感到迷茫了。

Mary 的故事是从时间的角度上来考虑的，下面这个例子提供了另外一种规划的方法。

“蔡佳娃，怎么样？从这个故事中得出什么启发了吗？”

“呃，故事倒是听明白了，只不过还没想清楚有啥含义。”

“哈哈，这其实是一个喜剧里面的笑话，不过其中的道理却是很值得我们学习啊！”

“那到底是什么道理呢？”

“我再给你出道数学题，不等式的证明：

求证  $\sqrt{3} + \sqrt{6} < 2\sqrt{5}$ ”

“啊？师兄你猛地一提我还真不知道怎么说。”

“还是我来说吧，解答如下：

因为  $\sqrt{3} + \sqrt{6}$  和  $2\sqrt{5}$  都是正数，

要证  $\sqrt{3} + \sqrt{6} < 2\sqrt{5}$ ，只需证  $(\sqrt{3} + \sqrt{6})^2 < (2\sqrt{5})^2$ ，只需证  $9 + 6\sqrt{2} < 20$ ，

只需证  $6\sqrt{2} < 11$  即  $\sqrt{72} < \sqrt{121}$ 。所以得证。”

“怎么样？看出点什么了吗？”

仔细分析解答这个证明题的思路：要想证明一个结果，先找出最终结果的充分条件，找到后把它当作一个新的需证结果，再寻找其充分条件，如此循环，直至最后的需证结果是一个无须充分条件证明的公理，或是可以通过已知求得的定理。这样，问题就解决了。

实现目标有的时候就像解答这样的证明题一样，定下了一个很明确的目标，但是现在却不知道从哪入手来实现目标，那么就可以从实现目标的那天进行回溯，推导出每件事的因果关系，最后就明白自己当前该干什么了！

“师兄，我好像看出点门道了，这个数学题和前面的故事都是从结果往前推的是吧？”

“正解，所以我们在规划目标的时候如果对于当前该干什么很是茫然，就可以用这种方法来让自己更清楚一些。”

“哦，师兄，我开始明白你的意思了。”

“嗯，非常正确。这下你应该不会再感到彷徨了吧？”

两则故事所阐述的道理各有侧重，第一个故事主要将未来到现在的时间划分成不同的阶段，而第二个故事注重于实现目标的各个步骤间的因果关系。但是两个故事都是按照从终点到起点的顺序对目标进行分析的，这才是最重要的一点。

对于那些有目标却很彷徨的人来说，故事中介绍的由未来及现在、由果及因的思考方式很值得借鉴，比如一个从事 Java 开发的人可以为自己的职业生涯做一个规划表，如表 3-1 所示。

表 3-1 职业生涯规划表

工作年限	需达到的目标	所要求
5 年	项目经理	需要对此行业领域的业务流程十分熟悉，项目架构能力很强
4 年	年薪 8 万	在公司需要有一定的不可或缺性
3 年	Team Leader	某个领域需要有很丰富的经验，需要技术能力水平很高
1 年	站稳脚跟	掌握流行技术，能适应开发这个行业
学校	顺利毕业，准备求职	需要通过学校考试，通过 SCJP 认证，需要有项目经验

细心的读者会发现，这个表是从上往下填写的。如此一来，就不会再为了自己清晰的目标感慨迷茫的现在了。在所要达到的目标旁边是达到该目标的要求，这样面对当下就不会无事可做。不过需要记住的是不要让目标只存在于确定的瞬间，定下长期目标并将其分为各个阶段，持续为之努力，收获的将是与汗水等价的成功与喜悦。

### 3.3 IT 认证的问题

曾经有个笑话说现在的社会已经进入了“纸器时代”，即人们所接触和追求的东西都是纸做的，如户口本、金钱、房产证、学位证等。在校大学生在为自己备战求职的时候，也免不了去寻找厉害的“纸器”来武装自己。证书，应该是大学生最能拿出手的纸制武器了。

#### 3.3.1 认证那点事

通过学校教育获得的毕业证书、学位证书，应该是最常见的一种证书。对于 IT 专业的学生来

说，除了这种学校的文凭证书，现在的 IT 行业还有种类繁多的各种认证，国内的如计算机等级考试、软件工程师等，国外的如思科的 CCNA、Sun 的 SCJP 等。

“师兄啊，我们现在有些同学都在忙着考证，我是不是也该考个证壮壮胆啊？”

“你的同学们都准备考什么证啊？”

“很多种呢，有的在准备国家的软件考试，有的在准备思科的 CCNA，还有的打算考 Sun 的 SCJP……这么多的认证，搞得我也很茫然。这个铺天盖地的证书到底在我们求职的路上扮演什么样的角色呢？是不是必须拥有的呢？”

“呵呵，你的困惑不无道理，证书目前在 IT 职场上确实有着举足轻重的作用，单看当下如火如荼的各种认证培训就可以明白。那么在回答你的问题之前，我先给你讲讲认证是如何在中国的 IT 行业中诞生和发展的吧。”

### 1. 认证的作用

“首先，我们先来谈谈 IT 认证的作用。一句话，IT 认证是作为进入职场的敲门砖或通行证而存在的，在这一点上和学校颁发的文凭证书是没有太大区别的。”

“那也就是说 IT 认证的生命周期也就是求职那一场恶战呗。”

“一旦进入了 IT 职场之后，手中的认证就基本失去了作用，就像买完票进电影院后票就已经作废了一样。”

既然 IT 认证的主要作用在求职这道关，那么其存在的价值之一就是弥补学校文凭在体现个人能力上的不足。而且因为目前很多不同领域的 IT 认证都是该领域的技术巨头公司组织开设的，所以一定程度上要比学校的文凭证书有更多的机会进入该领域的公司。

当然学校的文凭还是不可或缺的，至少它可以让你挺着胸脯说：“我是大学生！”。实际也是如此，IT 公司在招贤纳士时首先要看是否为大学生，不是大学生就基本没有任何机会了。

### 2. 认证出现的原因

“根据刚才说的 IT 认证的作用不难推断出来，IT 认证出现的原因肯定是和用人单位的招贤纳士的方式和标准息息相关。国内的 IT 认证出现于 20 世纪 90 年代，由于改革开放后教育产业的发展，人才与职场的供求关系已经发生了很大的变化。”

“师兄，那之前是什么样的职场现状啊？”

“在那之前很多大学还是负责分配工作的，就算是需要自己找也只是写一份简历即可，因为很少会有人会被拒绝的。”

“看来做一个 90 年代的大学生很幸福啊！”

不过，好景不长，慢慢人才的竞争开始激烈起来，学历开始成为区别人才水平高低的分水岭。而随着发展的继续，单单靠学历已经不能够保证找到一份满意的工作了，能力开始作为用人单位衡量人才的最重要标准。

在用人单位对求职者能力重视程度不断加大的同时，相比较于学历，能力作为一个在当时很难判断的指标也开始越来越困扰着每个 IT 公司。而随后出现的 IT 认证则开始在一定程度上成为了求职者的能力水平的表征，此时认证的含金量是极高的。

### 3. 认证的诞生和发展

“师兄，既然 IT 认证出现的原因是用人单位对于求职者学历和能力的重视程度发生了变化，那么它是如何诞生的呢？”

“首先在中国竖起 IT 认证大旗的就是响当当的微软帝国，眼光独到的微软在 90 年代末第一个在中国开设微软技术认证 MCSE 的培训，从此就拉开了认证培训大发展的帷幕。”

“那微软认证的效果如何啊？”

“那时是相当火爆啊，那个时候报名参加培训的人特别多。当时的微软 MCSE 讲师非常辛苦，必须每天喝中药护理嗓子，因为每天从早到晚不间断地有培训工作。”

IT 认证从诞生到现在，一直作为在一定程度上代表能力的衡量标准而存在，很多时候用人单位对于认证的青睐甚至都超过了学历。因为 IT 认证考试中的评判标准都是由企业来制定的，其技术含量比起学校颁发的学历，更能和业内的行情保持同步。

在微软的带动下，很多公司认识到设立自己的 IT 认证对于招聘人才和公司发展的重要性，于是 Sun、Oracle、RedHat 等 IT 巨头们也纷纷推出自己的 IT 认证。而国内由国家的有关部门牵头，也设立了很多国家级的 IT 认证，这些在后面小节中都会为读者介绍。

#### 3.3.2 现在的认证

中国市场上的 IT 认证发展到今天，认证种类和体系多种多样，含金量也高低不同，本节就来简单介绍一下行业中几种主要的认证。

“师兄啊，你刚刚给我讲了认证的作用和发展史，我都听明白了，接下来你还是讲讲我最关心的吧！现在 IT 行业主流的认证都有哪些啊？”

“呵呵，好吧，下面我就谈谈国内目前存在的国际认证和国内认证，让你有个比较深入的了解，这样如果你有打算的话就可以选一个比较厚重的敲门砖。”

##### 1. 微软

微软打响了我国 IT 认证的第一枪，所以微软的 IT 认证是非常有价值的。在技术不断发展的同时，微软的 IT 认证也在不断地发展变化。现在微软的 IT 认证种类有很多，不再是最初的 MCSE 和 MCS D 那样的绝代双骄了，主要项目如下所示。

- MCSE (Microsoft Certified System Engineer)

微软认证的工程师，这是微软传统的认证项目，一个合格的 MCSE 能够在 Windows 平台下进行软件或服务的部署、各种服务器的配置和数据库的管理等。所以要想考过 MCSE 必须要掌握 Windows 的网络及操作系统知识和 SQL Server 的配置及管理，以及其他 Windows 网络服务器组件的管理。

- MCS D (Microsoft Certified Solution Developer)

微软认证的解决方案开发人员，通过这种认证的技术人员可以在 Windows 平台下进行主要是面向企业级的程序开发。这种认证侧重于程序开发，但是并不强调具体的编程语言。通过这种认证需要对项目解决方案方面的知识非常熟悉，同时也要适当了解桌面应用程序和企业级应用程序的开发模式。

- MCDBA (Microsoft Certified Database Administrator)

微软认证的数据库管理员，通过这种认证的技术人员，可以认为其对于主要是 SQL Server 系列的数据库有比较深的了解，并且能够熟练对其进行开发和管理。通过这种认证需要对微软平台下的 SQL Server 有很强的驾驭能力，而且由于 MCDBA 面向的领域较窄，只涵盖数据库知识，所以很多情况下 MCDBA 都用来和其他认证如 MCSD 等一起来考。

- MCAD (Microsoft Certified Application Developer)

微软认证的应用程序开发人员，这是达到 MCSD 之前的一个初级阶段，通过这种认证的技术人员，能够熟练地开发 .NET Framework 下的应用程序。要想通过这种认证，必须掌握微软平台下的 Web 开发技术，包括服务端和客户端。

- MCDST (Microsoft Certified Desktop Support Technician)

微软认证的桌面开发人员。这种认证是随着桌面操作系统的发展而产生的，同时也能够弥补微软在 PC 端认证的空白 (MCSE 和 MCSD 均是面向企业级的)，通过此认证考试的技术人员能够为 Windows XP 及以上的用户提供桌面系统的技术支持和系统故障的排除等服务。

考试费用：约 400 元/科。不同的认证由不同的科目组成。

纯金指数：★★★★☆

以上给出的只是微软几种比较常见的 IT 认证，微软的 IT 认证种类繁多，而且随着技术的变化也在不断地改变，每年都会有认证被取消不再使用，有的证书每年还必须进行再次审核发放。因此，建议有意向报考微软 IT 认证的读者在报考之前了解一下微软最新的认证政策。

## 2. Oracle

Oracle 是目前仅次于微软世界排名第二的软件公司，主要从事数据库软件的业务。其主要面向的是大型的企业级数据库应用，同时也兼顾企业级应用软件如 ERP 和 CRM 等的开发。Oracle 推出的认证有 OCP 和 OCA，OCA 一般作为 OCP 的一部分来考。

OCP (Oracle Certified Professional)，甲骨文认证专家，通过此认证的人，将具有对 Oracle 各版本数据库进行熟练管理和设计开发大型应用的能力。Oracle 在大型数据库软件市场上占有绝对的优势，所以 Oracle OCP 认证的含金量很高。

考试费用：10000 元左右

纯金指数：★★★★★

Oracle 认证的高含金量也体现在其高额的考试费用中，不同的版本考试费用是不一样的，Oracle 10g 的费用就比 Oracle 9i 要低廉许多。

## 3. RedHat

RedHat 认证是 Linux 操作系统中的权威认证之一，在 Windows 世界微软的认证是一家独大的，而在 Linux 世界存在的很多不同系统版本的认证中，RedHat 的 RHCE 认证具有明显的统治地位。很多其他的 Linux 版本如 TurboLinux 等都以 RedHat 的认证作为标准。

RHCE (Red Hat Certified Engineer，红帽认证工程师)，通过此认证将具有对 RedHat Linux 操作系统的安装、配置及管理以及故障排除的能力，有点类似于 Windows 平台下的 MCSE。



RHCE 考试分为三个部分：第一部分是上机操作进行故障排除 (Debug Exam)；第二部分是多项选择题 (Multiple Choice Exam)，主要考 Linux 系统的各种命令；第三部分是服务器安装和网络服务配置测试 (Server Install and Network Service Setup Exam)，考生要根据要求对 Linux 系统进行指定服务的安装和网络配置的测试，并最终给出报告。

考试费用：每个部分的费用大约是 2500 元，如果一次不过第二次考就会变成 1000 元。

纯金指数：★★★★☆

Linux 操作系统目前正在快速地壮大，而作为 Linux 认证中的佼佼者，RHCE 应该是每个有志于从事 Linux 开发的技术人员的一张最好的能力证明书。

#### 4. Cisco

作为国际上最大的互联网解决方案提供商之一，思科公司的认证在业界有很高的认可度，思科公司主要提供网络硬件如路由器交换机及网络软件如视频会议和防火墙等。思科公司的认证主要是面向路由器和交换机的配置与管理，按级别由低到高分为 CCNA、CCNP、CCIE。

- CCNA (Cisco Certified Network Associate)

思科认证的网络工程师，它是思科认证系列中最基础的一个。CCNA 的相关考点主要是网络的基础知识和思科低端路由器交换机的基本配置和管理。CCNA 也分很多方向，如 CCNA Voice (网络语言技术)、CCNA Wireless (无线技术) 和 CCNA Security (网络安全技术) 等。

- CCNP (Cisco Certified Network Professional)

思科认证的网络高级工程师，是思科认证系列中实用价值比较高的一种认证。作为思科的主要认证，CCNP 认证要求能够对网络节点在一定规模以上的局域网或广域网进行思科系列路由器和交换机的配置与使用。

- CCIE (Cisco Certified Internetwork Expert)

思科认证的互联网专家，是思科认证系列中最高级的一种，也是国际互联网技术中最有权威的认证之一。CCIE 认证分为路由交换认证和网络安全认证等几个类别，是思科系列中最难考的一种认证，因此其含金量也是最高的。

考试费用：CCNA 和 CCNP 的考试分为考试费和资料费，资料费大概在 1000 元左右，考试费中每一门的费用大约是 1500 元。CCIE 考试分为资格考试和实验考试，前者费用大概在 3000 元左右，后者在 13000 元左右。

纯金指数：★★★★☆

思科作为互联网市场的领军人物，其认证的含金量自然不在话下。在路由器交换机领域，思科的认证还是有相当强的说服力的。

#### 5. Sun

Sun 公司的主要认证的就是针对 Java 技术与 Solaris 操作系统的，其中关于 Java 技术的认证保有量比较大。同时，由于 Sun 是 Java 技术的发明者，所以其在 Java 认证领域有着非常高的决定权。Sun 公司按照 Java 技术的层次高低和面向领域的不同将 Java 认证分为 SCJA、SCJP、SCJD、SCWCD、SCBCD 等八种认证，其相对关系如图 3-2 所示。

- SCJA (Sun Certified Java Associate)

Sun 认证的 Java 助理程序员，它比较简单，面向基础。主要考查的是对 Java 基本语法的掌握，所以含金量不是很高，考量不是很大。

- SCJP (Sun Certificated Java Programmer)

Sun 认证 Java 程序员，是 Java 认证里面最具有代表性的，同时也是人气最高的一种认证。SCJP 要求考生掌握 Java 技术的核心知识，并熟悉常用的 API 用法。SCJP 通常作为 Java 开发人员步入职场的一个敲门认证，有较高的含金量。

- SCJD (Sun Certified Java Developer)

Sun 认证 Java 开发人员，此认证需要首先通过 SCJP 认证才可以报考。SCJD 认证考试分为项目开发部分和项目论述部分，在第一个部分会被要求开发一个指定应用的设计方案及实现，然后在第二部分对其进行讲解和论述。

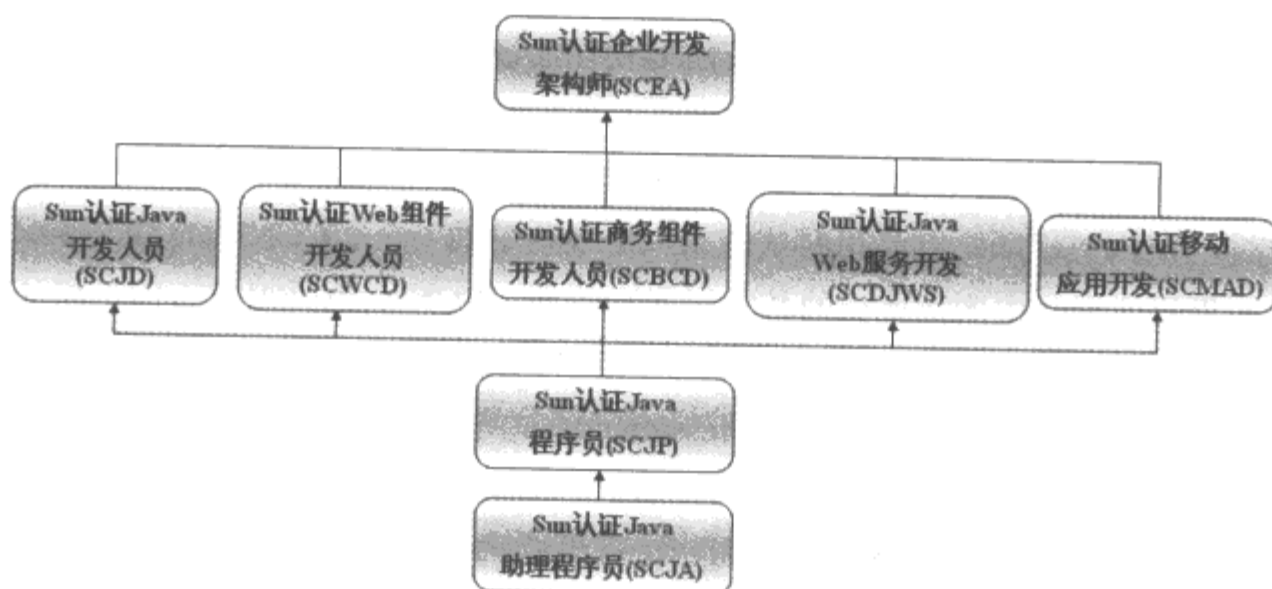


图 3-2 各种 Java 认证之间的关系

第一部分时间很宽裕，可以自己在学习工作之余慢慢完成，完成后提交即可。而第二部分则不同，是以面试答辩的方式进行的，而且要求使用英文，有一定难度。

- SCWCD (Sun Certified Web Component Developer)

Sun 认证的 Web 组件开发人员，此认证也需要首先通过 SCJP 认证才可以报考。此认证主要考查使用 Java 技术开发 Web 应用的能力，需要的知识涉及 JSP、Servlet、JavaBean 等 Web 开发相关的内容。

- SCBCD (Sun Certified Business Component Developer)

Sun 认证的商业组件开发人员，此认证同样需要首先通过 SCJP 认证才可以报考，主要考查的内容是开发部署 EJB 组件的相关知识。

- SCDJWS (Sun Certified Developer For Java Web Services)

Sun 认证的 Web 服务开发人员，此认证也需要首先通过 SCJP 认证才可以报考，此认证和 SCWCD 的区别就是对 Web 服务程序的设计有较高的要求。

- SCMAD (Sun Certified Mobile Application Developer)

Sun 认证的移动开发人员，此认证同样需要首先通过 SCJP 认证才可以报考。SCMAD 主要的考试范围是 Java 在移动设备如手机或嵌入式设备中的应用开发，主要用到的知识是 Java ME 领域的。

- SCEA (Sun Certified Enterprise Architect)

Sun 认证的企业级架构师，此认证是 Sun 认证中级别最高的认证，而且不要求通过前面的认证即可报考。SCEA 的考试分为三个阶段：第一阶段进行架构基础知识的测验；第二个阶段要求进行一个实际项目的架构设计；第三个阶段需要针对第二个阶段设计的架构进行讲解和分析，也就是面试答辩。

Sun 认证的考试全年都可以报考，只要在考试之前到授权考试中心进行预约即可。与其他国际 IT 认证相同，Sun 的认证考试大部分都是英文版的，因此对专业英语水平也有一定的要求。

考试费用：Sun 的认证考试每年都会有优惠期，其他情况下大概都是 1500 元左右。

纯金指数：★★★★☆

## 6. 国内 IT 认证

国内的 IT 认证是在国际 IT 认证的迅速流行之下开始的，起步较国际 IT 认证晚。国内的各种认证一般都是由劳动部、工信部（原信息产业部）、教育部来颁发的，除了那些和国际厂商类似的表征能力的认证之外，还有一些认可度比较高的如律师资格证、教师证等上岗证书。

国内的 IT 认证主要有计算机技术与软件专业技术资格考试，软考是国家级的 IT 专业技术人员从业资格考试，分为软件设计和网络管理两个大类别，每个类别划分为不同的等级，分为技术员、助理工程师、工程师、高级工程师。

国内的 IT 认证既是职业资格考试，也是水平资格考试，同时还具有职称性质，而且国内的 IT 认证可以在日本、韩国与某些认证互相承认。

考试费用：国内 IT 认证的考试费用根据不同等级收取不同的考试费，一般来说为 100 元到 300 元左右，同时因城市的不同而略有差异。

纯金指数：★★★☆☆

国内的 IT 认证在国内有一定的肯定度，但是总体和国际的 IT 认证比起来，专业性要差一些，类别的划分上也不够细致。



**提示** 本书中列举的各种认证仅供参考，具体认证政策需以最新的官方消息为准。

### 3.3.3 该不该考个证

“师兄啊，听你说了这么多 IT 的认证的内容，那么对于我们这些一两年后就要杀进职场的人来说，该不该搞一个证书呢？搞个什么样的呢？”

“就现在来说，证书并不能绝对代表一个人的能力，因为现在的用人单位比以前聪明多了，通过笔试面试这几关就能大致了解一个人能力水平，而且很多情况下用人单位对于求职者的技能要求也不仅限于 IT 认证所包含的内容。”

需要说明的是，尽管 IT 认证的作用已经不像前些年那样具有决定性，对于即将踏上求职路的 IT 新手也并不是不可或缺的必备品。但是拥有一张漂亮的 IT 证书，还是很能为自己的形象加分的。尤其是拥有一张高分的认证成绩单，那将会让招聘单位刮目相看。

在某些情况下，IT 认证在求职的过程中所起的作用还是非常大的，因为 IT 认证都是由企业带头设立的，其技术特点和所依附的企业有很大关系。有时候同一类型不同公司的 IT 认证可能所

需求的知识是千差万别的，如 Cisco 的网络工程师可能就无法通过华为的网络工程师认证。

因此，遇到一些对技术要求专精的岗位，如数据库管理员、网络工程师等，最好还是有一证在手，因为那种 IT 认证对于个人能力水平的代表性很高。

“师兄，听了你这么一说，我想我还是以后考一个证书比较好，而且最好是拿到高分，这样的话或多或少也会为自己的技能增加一点说服力，是吧？”

“嗯，说得没错。没有证书不代表吃不上饭，但是有证书或许能保证你早些吃到。”

“那师兄，我应该选择哪种 IT 认证比较好呢？国际的还是国内的呢？”

“我建议你还是选择国际的 IT 认证比较好，因为国际的 IT 认证比较难考，含金量也比较高，考下来比较有成就感。同时国际 IT 认证的考试费用也是相当高；而国内的 IT 认证就比较大众化，在专业程度和知识深度上都比国际 IT 认证要欠缺一些。”

总体来说，国外的 IT 认证发展比较早，其划分的体系也更加完善，所囊括的技术知识也比较先进全面。同时由于国外做 IT 认证的都是那些在相应领域一言九鼎的 IT 巨头，相比国内的来说要更加高端和专业，所以其认证也自然要显得雍容华贵一些了。

国内的 IT 认证则相对来说门槛低，更容易通过，在技术水平上没有国际 IT 认证的代表性高，这也使得其权威性要稍差一些。最重要的一点，国际的 IT 认证在很多地方都有效，认可度很高；而国内的 IT 认证则含金量不够高，只能在国内使用。

### 3.4 本章小结

IT 这个江湖令人向往也令人畏惧，想要从学校的深山中走出来，顺利地进入 IT 江湖，没有充分的准备是不行的。希望各位读者珍惜自己在山中的时间，为了心中的目标暗自努力，深学苦练，争取让自己的下山之路平坦顺利。

# 第 4 章 必须通关的游戏——求职之旅

如果把找工作这个必经之路比做打游戏的话，它与真正游戏的主要区别就是必须通关。否则不论打到第几关都算输，没有水平高低，只有成败。而这个游戏，除了极少的特殊情况，绝大多数的求职者都是按照投简历、笔试、面试、试用期这条路求职成功，最后转正成为正式员工。

就像游戏也会有 bug 和秘籍一样，任何竞赛性质的活动都是有对应技巧的。本章旨在向那些摩拳擦掌一腔热情、豪气冲云天的同学们介绍每个环节的通关攻略。同时，也会向那些一咬牙一跺脚气急败坏决心造假的读者们分析一下每一关卡的蒙混几率。不过这些所谓蒙混几率就像所有方便面包袋上的效果图一样，旁边必有一行小字：“仅供参考”。

就像前几章一样，我们憨厚勤奋的主人公蔡佳娃同学一直在师兄牛开复的指点下朝着一个 IT 牛人的目标稳步前进。现如今蔡佳娃已经升入大四下半学期，初出茅庐的他就要信心满满地开始求职之旅了。让我们追随他的脚步，看看现今的职场风云吧。

**提示** 在这里要提一下另一个求职区别于打游戏的特点，那就是虽然你可以失败了重新来过，但是切记，关卡已经变了，上次失败的经验下一次可能并不会有用。出于讲解的需要，我们的蔡佳娃在这本书中是一粒蒸不烂、煮不熟、捶不扁、炒不爆、响当当的铜豌豆，屡战屡败，屡败屡战。


## 4.1 简历靓起来

好的开始是成功的一半，只有简历被看中才会有资格进行笔试和面试，才算真正进入了游戏。同时这一关也是比较困难的，因为这一关玩家太多，数据的真实性最不可靠。而且就算一个求职者于情于理都应该通过，如果出现简历书写上的缺憾，那也只能仰天长叹，从头再来。

### 4.1.1 简历不是这样写的

蔡佳娃目前对于工作这件事还并不是很着急，只是准备提前感受一下这个过程。即便如此，在满怀信心地投出第一份简历后，却是泥牛入海，毫无消息。这还是让蔡佳娃有点小郁闷，下面请看蔡佳娃简历的第一版。

个人简历	
I 个人信息	
姓 名：	蔡佳娃
性 别：	男
出生年月：	xxxx年x月xx日
籍 贯：	xxx省xx市





毕业院校: xx理工大学

学 历: 大学本科

专 业: 计算机科学与技术

联系电话: xxxxxxxx

E-mail: xxxx@xxx.xx

## II 自我评价

待人真诚, 对待工作认真负责, 有团队意识, 能吃苦

## VI 专业技能

学过数据结构、操作系统、计算机原理、计算机体系结构

学过 IBM-PC 汇编语言程序设计、C 语言程序设计、C++ 程序设计、Java 网络编程

学过数值分析、网络协议、电路、数字逻辑、离散数学、大学物理、计算机体系结构

学过高等数学、大学英语, 熟悉 HTML/CSS/JavaScript

**专家分析:** 这份简历放到考官面前会被直接“秒杀”, 原因是什么呢?

- 专业技能中列出了所有在大学中学过的课程, 这就是被“秒杀”的主要原因。考官通过简历要查看的是应聘者的专业技能, 而不是应聘者所在学校的教学计划。
- 但凡对自己能力自信的学生, 都会优先地将自己的特长和精通的技能写到简历中, 根本不会为学校的课程留有大量的篇幅。

因此, 这种简历遭到舍弃也就无可非议了。

蔡佳娃是个好琢磨的人, 他略加研究, 觉得自己的专业技能过于小儿科, 不够时尚, 搜肠刮肚般回顾了一下自己在大学里学过的技术, 便又推出了第二版简历。

# 个人简历

## I 个人信息

姓 名: 蔡佳娃

性 别: 男

出生年月: xxxx年x月xx日

籍 贯: xxx省xx市

毕业院校: xx理工大学

学 历: 大学本科

专 业: 计算机科学与技术

联系电话: xxxxxxxx

E-mail: xxxx@xxx.xx



## II 自我评价

待人真诚, 对待工作认真负责, 有团队意识, 能吃苦

## VI 专业技能

精通 C/C++、VB、HTML、CSS、JavaScript

精通 .NET、C#，熟练使用 SQL Server

精通 Java SE 类库、Java EE、Java ME、JSP、Servlet

熟悉 Linux、Solaris 等操作系统

精通 Photoshop、Flash，熟悉 EDA，熟练掌握 C51 单片机

这个看似非常完美的简历应该不会有什麼纰漏了吧，蔡佳娃这样想着，就满怀信心地开始了第二次的简历大投送。在经过了了好几天的寂静，又接了两个拒绝自己的电话之后，蔡佳娃不得不开始再次审视自己的“完美”简历。

**专家分析：**第二版简历看起来很漂亮，为什么会无人问津呢？

- 这份简历水分很大，估计很多考官会这么想。因为所有 IT 业界的“过来人”都明白，短短四年的大学时间，是不可能精通这么多专业技能的。应聘者明显是在夸大事实，不够诚实。
- 此类简历也暴露出应聘者对自己的不了解和不自信，学过这么多技能，居然不知道自己最擅长的是什么。或者觉得自己没有出色的地方，只好实行平均主义，“精通”了所有的技能。

于是乎，这种“水货”简历也难免遭到“秒杀”。

蔡佳娃犹豫了一下，决定去掉简历上一些出镜率太高的“精通”，于是“精通 Photoshop”、“精通 .NET”、“精通 Flash”等被删掉了。

这么一来，蔡佳娃心里的惭愧感才少了一些。毕竟他只用 Photoshop 将图片的背景处理成透明，Flash 也只是搞出过地球围绕太阳转的引导动画。而至于 .NET、VB 等，如果看过书就叫做精通的话，那对于自己真正学过、研究过、思考过的 Java 他应该用哪个词来修饰呢？

如此一来，蔡佳娃便一不做二不休，大方地删掉了“熟悉 EDA”、“精通 C51 单片机”……

这么一来，蔡佳娃的简历干净了许多。专业技能里面只剩下“精通 Java 类库”、“精通 Java ME”、“精通 JSP、Servlet、HTML、CSS、JavaScript”等与 Java 有关的内容。浏览一遍，蔡佳娃又在个人评价中加上了一句“热爱学习”，然后直奔复印店。

不用说，第三版简历仍然还是泥牛入海，没有消息。

**专家分析：**虽然蔡佳娃一直在改进，但是第三版简历还是有欠缺的地方。

- 第三版简历乍一看没有什么问题，可是专业技能还是不够具体，比如 Java EE 这方面如果加上“熟悉 WebWork2、Spring、iBatis、Lucene 等开源框架的使用”和“熟练使用 SQL，熟悉 Oracle、DB2 等大型数据库下的 SQL 开发”等更加具体的内容，考官便会对应聘者比较丰富的知识留下一个好的印象。
- 另外，蔡佳娃最后加上的一句“热爱学习”也会为他减分不少，因为用人单位更愿意招聘一个来到公司就能立刻上前线为公司创造财富的成手，而不是一个需要先消耗单位资源学习一通才能胜任工作的员工。

其实，蔡佳娃的三份简历一直得不到用人单位的肯定，很重要的一个原因就是没有主旨。简历和文学创作一样，需要一个主旨，然后所有自己添加上的知识、技能和经验都为一个主旨服务。这个问题在写简历的时候就应该弄明白，而不要等着考官来研究你简历的中心思想。

### 4.1.2 写出出色的简历

上一小节我们初识了简历关的困难，在“欣赏”了一下蔡佳娃同学的简历之后，也分析了其失败的原因。写出一份让人眼前一亮的简历对于打响冲锋职场的第一枪十分有必要。请看蔡佳娃是如何在高人的指点下重返战场，勇闯简历关的。

简历上的失败让蔡佳娃很是郁闷，不得不求助于刚出差回来的师兄牛开复。

困惑的蔡佳娃带着自己简历的三个版本和最近尚未问世的第四版找到了师兄牛开复。牛开复看过简历，把蔡佳娃狠批一顿，接着便向毫无头绪的蔡佳娃讲了一些自己的求职心得。

“蔡佳娃，我当时找工作从来都不是一份简历投到死，每次投简历之前都要先了解一下用人单位的岗位需求，尽量让自己的简历向那个方向靠拢。这样全神贯注地射出一箭，比起漫无目的地乱开枪命中率要大得多。”

“师兄，我们宿舍有的人简历做得很精美，还放进去几张艺术照片……”

简历越是豪华，说明里面真正有价值的东西越少，不然不会花大手笔做表面文章。往往这种简历用人单位是很嗤之以鼻的。写这种简历的人，估计对 IT 这个行业了解太少，IT 开发行业是研究技术的，要求的是踏实的态度和严谨的作风。这种华而不实的简历不仅让人看着反感，也不实用。”

需要说明的是，作为拉动全球互联网技术前进的发动机，IT 行业在招聘人才方面越来越多地通过网上来进行，尤其是简历关，很多简历都是在网上传送的电子版。所以各位求职者在备战简历的时候也要为自己多开辟一个网上战场。

“师兄，那写简历的主要原则是什么呢？”

“简历不在字数多，篇幅大的简历反而让考官觉得很烦琐，同时也会淹没你的亮点。所以写简历就是要简单明了，旨在向用人单位展示你对于所招聘岗位的合适程度。”

“那简历的主要部分就是专业技能了？”

“也不全对，你的这几份简历有个共同的毛病，就是都没有写自己的项目经验，就像我之前对你说的，项目经验也是用人单位衡量一个人能力水平的标准，考官是综合你的专业特长和项目经验来判断你的能力是否能够满足他们的岗位需求的。”

“哦，那我回去把我之前做的一些项目加上，我的项目经验不是很多，可不可以编造一些加上啊？”

对于大学生来说，项目经验是个让人又爱又恨的东西。现实的情况是：大学生不仅要有一定的项目经验，而且必须是真实的项目经验，厉害的考官一眼就可以看出来简历上的项目经验是真是假，是大是小，是实是虚。套句古语就是：成也经验，败也经验。

就算在简历关能够用不真实的项目经验侥幸过关，但简历之后的笔试、面试，以及随后的试用期，都是在考查你这个人有没有撒谎，是不是像简历上说的那样有能力。

虽然很多人在写简历的时候都会稍微地把自己夸大一些，这或许已经是个心照不宣的潜规则，因为大家都夸大一些，如果你不夸大，你就是落后了，但是要注意这个度，否则，或者会被发现欺瞒而被淘汰，或者成功过关但到最后难免原形毕露。

“师兄，那我还是老老实实说真话吧。其他部分呢？比如自我评价、兴趣爱好？”

“这些东西用人单位并不是特别关心，因为你的专业技能、项目经验都可以通过笔试、面试或试用期进行检验，而你的性格、爱好就不是那么容易试出来。况且相对于专业技能和项目经验，你的性格爱好对于为公司创造财富并不是最重要的，基本上是属于比较虚的东西。”

“呃，师兄啊，听了你这一席话，我倒是明白了不少。可是，对简历的整体把握上还是有些理不清的地方。”

“这哪是理不清啊，你就是懒得重新造，想有一个模板依葫芦画瓢是吧？”

“嘿嘿，还是师兄了解我啊。”

“好吧好吧，我把我以前找工作的一份简历给你参考参考，你大概按照这个样式和层次来制作你的简历，记住我说的简单明了就好。”

“谢谢师兄，师兄我对你的敬仰犹如滔滔江水，连绵不绝，又犹如黄河泛滥，一发不可收拾……”

“行了行了，别贫了。这回你要再不成功，你就别回来见我了。”

蔡佳娃拿到牛开复师兄的简历，的确是有种豁然开朗的感觉。

## 个人简历

### I 个人信息

姓 名：牛开复

性 别：男

出生年月：xxxx年x月xx日

籍 贯：xxx省xx市

毕业院校：xx理工大学

学 历：大学本科

专 业：计算机科学与技术

联系电话：xxxxxxxx

E-mail: xxxx@xxx.xx



### II 自我评价

专业技能扎实、吃苦耐劳、富有钻研精神、团队意识良好。

擅长 J2EE 开发，熟练掌握 Lucene、WebWork2、iBATIS、Spring 等开源产品，熟练掌握 Web Service、Socket、HTTP 等通信接口的应用。

### III 项目经验

2006.10—2007.06 信息产业部通信软件工程中心项目的开发

项目描述：移动基站软件项目

项目职责：在研发部担任软件工程师，负责用 Java 开发移动基站软件

2007.08—2007.09 电子商务网站积分、手机话费支付接口

项目描述：E 动商网与湖北移动的电子商务网站积分与小额话费的接口开发，该接口使用 Web Service 与 Socket，主要供网站的展现层查询和业务调用。



项目中职责:

主要负责接口的设计和开发,包括结算算法的设计和实现、部分业务逻辑的封装等。

#### IV 获得奖项

2006 年 10 月—2006 年 12 月 参加由 Sun 公司举办的第二届 Java Cup 全国大学生信息技术大奖赛 (NetBeans Plugins Code Jam), 获得大赛三等奖。

#### V 认证情况

大学英语六级 613 分

2007 年获得了 SCJP 证书 成绩 93 分

#### VI 专业技能

- (1) 熟练使用 Java SE 核心 API 及 JavaScript 语言。
- (2) J2EE 服务器: 精通 Weblogic、JBoss 中项目的部署与运行。
- (3) 持久层: 精通 JPA、Hibernate 的开发与调试, 有实际项目工作经验。
- (4) Web 层: 精通 JSP、Servlet、JavaBean、Struts、JSF 的开发与使用, 有实际项目工作经验。
- (5) 业务层: 精通会话 Bean 的开发与部署, 对 Spring 也有较深的研究。
- (6) 数据库: 精通 Oracle 数据库的使用、脚本的开发、数据库性能优化、分布式环境的搭建。另外精通 MySQL 的使用, 有实际项目工作经验。
- (7) 掌握 Linux 系统常用指令, 有在 Linux 下部署、维护 Java EE 项目的经验。

#### VII 兴趣爱好

篮球、足球、电脑游戏

蔡佳娃如获至宝, 参考着师兄牛开复的简历, 冷静地研究了一下用人单位的岗位需求, 写出了第五版简历。然后, 蔡佳娃将简历发给了自己之前分析过的几家比较合适的单位。

蔡佳娃并没有傻傻地等待回复, 因为牛开复师兄告诉他, 他更应该关心的是简历通过之后的第二关——笔试的问题。所以蔡佳娃一直在宿舍闭关研究 Java 技术。

过了两天, 蔡佳娃给师兄牛开复打电话: “师兄啊, 我接到了一个笔试通知!”

**专家分析:** 总体上讲, 蔡佳娃算是一个比较用功的同学, 不应该在简历关就挂掉。只是还没有学会如何驾驭自己的简历让其博得考官的好印象。其实就像故事里牛开复师兄说的那样, 好的简历大概遵循如下几个原则:

- 简明扼要

简明即指简历内容简单明了, 相比较于洋洋洒洒篇幅很长的简历, 考官更容易在精简朴素的简历中快速发现应聘者的亮点。扼要即指主题明确, 能让考官在看到简历后就很容易根据应聘者的能力为其做一个定位。

- 匹配度高

简历信息与招聘单位岗位描述匹配度高, 就像牛开复师兄说的一样, 简历不应该通用一份。根据招聘单位的职位描述, 应聘者也要适当调整, 力求自己的简历所描述的技能与单位的岗位描述相匹配。



- 关键信息突出

关键信息突出，简历内容的顺序不需要依照传统，可以把需求相关度高的内容放到上面，比如在应聘一家对 Java SE 要求比较高的单位时，可以把自己 SCJP 的高分成绩放到个人信息的下面，然后再继续介绍自己的技能特长等。

### 4.1.3 如果是机器筛选简历

IT 行业在不断地改变着每个人的生活，最主要的表现就是把人从烦琐的劳动中解脱出来，或者至少是大大提高其效率。前面已经说到很多时候简历都是在网上交流，如今不仅如此，简历的初步筛选也开始渐渐由人力劳动转为由任劳任怨的机器搞定了。

“师兄啊，我们宿舍有个人，他的简历和我之前写的一份挺像的，在专业技能一栏里写了好多精通的技能。你不是说这种简历会被考官直接‘秒杀’吗？可是他和我一样都接到了笔试通知呢！”

“有这个可能啊，现在有些单位招聘人的时候是用机器来筛选简历的，机器虽然快，但机器是死的，永远也赶不上人脑聪明，只是按照固定的算法挑选出符合条件的简历。”

“那岂不是会有许多冤假错案了？”

“冤假错案倒是不大可能会有，更多的应该是漏网之鱼吧。因为机器筛选简历是根据简历中的内容与输入的招聘岗位需求关键字的匹配程度来进行的。所以只要把自己的专业特长写得全面一些，就可以骗过机器的眼睛了。”

“这种方法不太科学吧，大家如果都写完美简历，机器会让所有人都通过的。”

“非也，机器一般只是起到初步过滤的作用，用人单位会继续人力审查机器挑选出来的简历，如果发现钻空子的‘完美简历’，依然会直接‘秒杀’掉。你们宿舍的同学应该还是比较走运，不仅逃过了机器的眼睛，还骗过了考官，不过到了笔试应该就不会那么走运了。”

应聘求职者越来越多，简历也像洪水猛兽般涌向了用人单位，善于创新的 Google 在几年前美国总部招聘中就率先采用了机器过滤简历来筛选求职者。随着数量越来越多简历铺天盖地地飞向各个用人单位，机器过滤简历便逐渐成为各用人单位挑选人才的第一个大筛子。

但是考虑到筛选算法不可能完善到与人力筛选有相同的准确性，只是比较关键字的匹配程度，如果应聘者各方面能力都不错，但是表达方式有问题；或是应聘者能力并不是很出色，但是善于雕琢文字，都难免会发生故事里的“冤假错案”。

机器开始过滤简历或许并不是坏事，至少这个一丝不苟的考官会让应聘者不敢对自己的简历再有所怠慢。其实不论是面向机器还是人，在书写简历的时候都应该注意让自己的经验介绍和专业技能规范、清晰、醒目。

### 4.1.4 简历小结

通过前面几个小节介绍，相信某些读者可能会对简历这一关有所“想法”，会对于其公正性保守地表示怀疑，其实这一关的蒙混几率应该是很低的。根据笔者的经验，靠真功夫写出来的简历与迎合拼凑出来的简历还是有很大区别的。

这一关不仅会淘汰一些能力稍逊的应聘者，有时也会向有真才实学的应聘者宣布“Game

Over”。因为简历也是一个人表达能力、思考能力的体现。就算你胸中再有波澜，双手却无从掌墨，仍然是无法向考官要来第二关的入场券。



小小一份简历，是每个应聘者走向自己职业生涯的敲门砖。简历即是自己的门面，对自己做一个透彻的分析，定位自己的职业目标，从而写出一份简明扼要、关键信息明确的简历，为自己的职业生涯走向成功开个好头，不要让简历的内容和形式亏了自己的才能。

## 4.2 笔试，混可不行

通过了简历关的“海选”，笔试这一关算是比较回归校园了。从起源于隋朝的科举，到现在的学校教育，考试一直作为一种相对简单而又比较公正的选拔人才的方式。本节将着重介绍笔试的考试范围和如何准备笔试。同时，在上一节如愿通过简历关的蔡佳娃同学也要开始他的第二关历险了。

### 4.2.1 初识笔试

如果说简历关是时好时坏、发挥不稳定的主观题，那么笔试关就是真切实在、考你没商量的客观题了。关于考试，每位同学的临场经验都很丰富，不再需要有人指点。而笔试和普通考试除了在内容上存在不同之外，考场纪律也不会有校园考试那么严明。

到了笔试这步田地，还想要作弊可是需要很大勇气的，不仅是要求胆量，还要有勇气面对将来有可能被戳穿的局面。同时笔试中面对着满屋子都是来抢这个饭碗的竞争者们的虎视眈眈，作弊的手段也非常有限了。

蔡佳娃也深知笔试马虎不得，所以简历发出去后便一直闷在宿舍准备可能到来的笔试。结果刚闭关了两天，发现 Java 实在是太博大精深了，于是又求教于牛开复师兄。

“师兄，我前两天投出去了几份简历，有了你的指点，果然接到了一家公司的笔试通知，就开始研究笔试了，师兄你再开导开导我呗。”

“笔试是个实在的东西，没什么技巧，就看你平时的用功程度和知识积累了。”

“那笔试一般都考什么呢？Java 这块蛋糕这么大，我怕我没时间和精力彻底吃掉啊。”

“看看，又想偷懒了。好吧，我大概介绍一下笔试的内容。鉴于你这种临时抱佛脚的情况呢，我建议你下大力度好好啃一啃核心 Java，比如 Java 中的内存管理、多线程开发、集合框架等，这都是一些比较基础但又非常重要的部分，高端的内容还是少研究比较好。”

“哦，这样就简单许多了呢。”

其实，很多读者都会有蔡佳娃这样的想法。觉得 Java SE 比较容易掌握，只是基础而已，而 Java EE 和 Java ME 相对来说更值得花费工夫去研究。其实基础并不一定简单，就像建大楼时地基是最费事也是最费资金的，只不过由于平时不能直接看见它普通人不重视罢了，承建商可是很重视的啊。

另外，大部分人都错在把 Java 仅仅当作一门语言来学习，而不是在学习不断体会面向对象的思想以及 Java 本身的平台特性，读者朋友们在这些方面要多注意下工夫。

“简单？看来你对 Java 领悟不够高啊。我来考考你，面向对象的四大特性是什么？”

“这个简单，抽象、封装、继承、多态。”

“好，那什么是多态？”

“啊？？多态是……一个父类……派生出多个子类，然后子类调用……”

这是一个很多人都知道但是却无法说清楚的概念，多态的含义是不同的对象有相同的一般轮廓或形态，可以执行某个相同的操作，但是执行过程和细节却各不一样。比如所有的机动车都有刹车的功能，但是不同车的刹车方式和效果却截然不同。

很多 Java 初学者就跟蔡佳娃一样，研究问题只局限在表面，学习 Java 只注重语法格式，只把目标限定在熟练掌握循环嵌套、分支和判断等流程控制这些超基础的内容之上，结果遇到像多态性定义等类似的问题就彻底瘫痪了。

“哦，果然其中奥妙多多啊！”

“再问你一个，Overload 和 Override 的区别是什么？”

“一个是重载，一个是重写。重写是子类对于与父类同名的方法有不同的实现，重载和重写差不多，是名字相同但参数个数和类型不同的一组方法，重载也属于多态性吧？”

“不完全对，方法重载是可以提供类似多态性的好处，但是它与多态机制有本质的区别，重载一般指在一个类内部的方法重载，重载不同于覆盖，重载的方法是可以改变返回值类型的。最重要的是，重载不是面向对象专有的。”

“听君一席话，省我十本书啊！”

“说到书，我可以向你推荐一本讲核心 Java 的好书《Java SE 6.0 编程指南》，这本书讲得很全也很透彻，你可以好好读一读。”

“听说过，哪天拜读一下。这样按师兄你说的准备应该差不多了吧？”

**提示** 其实如果有读者心里太没底，可以在求职前去考一下本书在第 3 章提到过的 SCJP 认证，如果通过并且成绩在 85 分以上的话，笔试这一关中核心 Java 这部分过起来还是比较游刃有余的。但是，如果像蔡佳娃那样认为笔试只有这些，那就大错特错了。

“蔡佳娃，你就是有些毛躁，我话还没说完呢。笔试可不是只考你核心 Java 就了事的，不然大家都扬着 SCJP 的高分成绩单招摇过市了。根据用人单位的岗位需求，EE 和 ME 方面的知识还是要考的，但大多是一些描述性质的问题。”

“师兄啊，对于我们菜鸟，最好的说明手法是举例子，呵呵。”

“真服了你了，好吧。比方说问你 Java EE 是什么？”

“啊？？还真让师兄你说准了，往往常挂在嘴边的并不是我们彻头彻尾明白的，这个问题我还真没法回答呢。”

“所以说学习的时候钻得深是好事，但也总要不时的浮出水面把握一下全局。Java EE 是原 Sun 公司（现在被 Oracle 收购了）提出的多层（multi-tiered）、分布式（distributed）、基于组件（component-base）的企业级应用模型（enterprise application model）。”

“真是惭愧啊，这么久了才知道 Java EE 的身份唉。”

“像这样的题还有很多，比如解释 JNDI、JMS、SOAP、JTA、RMI、JDBC、EJB 技术，或者浅谈一下 Struts，描述一下 Servlet 的生命周期等。”

很多概念性问题都是这样，往往嘴上念得最顺溜的并不是我们最明白的，学习 Java 要不断地去思考和领悟，要通过 Java 这门语言本身感受到面向对象设计的思想。这是个自然而然潜移默化的过程，既不能过分强求于自身，也不要妄求转载自他人。

学习的结果全在自己的态度，如果把 Java 仅当作一门技术来学，那么 Java 也只能让你做一个手艺熟练的敲代码人员。但如果把 Java 当作一种面向对象编程的思想模式，那么 Java 也必将带领你进入一个更为广阔的程序开发天堂。

“哎，那师兄你说的这些知识点看看书应该好拿下吧！哈哈。”


“非也非也，有些东西是你看看就会记住的，有些东西却必须在有一定运用经验和领悟的基础之上才能牢记和掌握的。近期可能来不及了，你以后一定要注意总结思考啊。”

“嗯，师兄说得太有道理了！”

“哦，对了。还有个部分笔试也会考到，那就是数据库方面的知识，因为但凡一些实用的有一定规模的应用程序，没有不用到数据库的。但是这方面涉及较少，而且这方面知识的积累更多是来源于实践。临时挑灯夜战估计作用也不会很大，但是参考一些常用的笔试题还是很有裨益的。”

“好的，现在我的思路开始有些清晰了。师兄，我会好好准备的！”

既然是考试，肯定要有统考大纲，本节大概讨论了笔试的考试范围，即核心 Java、Java EE 或 Java ME 方面描述性的原理或知识、数据库方面的知识。这三个部分所占比重一般依次递减。有了统考大纲，备战时就大致有了方向，不过有些东西像核心 Java 可以更多依赖读书和体会，有些东西却是在每一次的编程实践中领悟到的。

 **提示** 请牢记我们为灵活运用 Java 而战，并非为熟悉 Java 而战，我们是实践主义者。前辈的经验告诉我们，只有理论联系实践才能真正开发出有价值的软件。

#### 4.2.2 牛刀初试

一个礼拜过去了，蔡佳娃同学果然还在专心备考。牛开复师兄过来看他了。

“小蔡啊，研究得怎么样了？”

“大后天就要去笔试了，所以我现在临阵磨枪，不快也光啊。”

“呵呵，光看书也不行，来，我给你找来了一些笔试考题，让你提前练练兵。”

“不是吧，我怕我还没准备好……”

“哪有什么准备好不好的啊！做做看先！”

像蔡佳娃这样的人就是有些过于谨慎了，很多时候，事情都不可能等到准备得万无一失才去做。比如排练节目，每次排练都会觉得有瑕疵，不完美，但等到硬着头皮上台演出的时候，舞台上是什么样的，在观众眼中，那就是最终版本，是经过准备后的最完美展现。什么时候亮剑出鞘，什么时候就算准备好了。自信，有时候也是一种胆量。

“那师兄你考考我吧，看看我能不能一剑封喉呢？”

“小子你还挺不谦虚，好，先给你介绍一下笔试的题型吧。笔试大概有选择题、简答题和编程题。有的时候还会有一些智力题，或者叫谜语。”

“哇，内容好丰富，还有编程题和智力题。”

“是啊，用人单位为了招聘优秀的员工可是想尽了办法呢。”

“那我们就开始吧！”

蔡佳娃接过师兄的考题，埋头做了起来。

**提示** 在准备笔试的过程中，可以适当地做一做笔试的经典考题。因为把书看懂了是一回事，能够在遇到问题时运用看懂的知识又是另外一回事。但是不建议大家把题目背下来，搞题海战术。因为抛去极小的成功几率不谈，本书倡导的可是一条光明正大的高手之路。

师兄看了看时间，说道：“差不多了，来，甭管做成什么样子，我们一起看看吧！”

蔡佳娃有些不甘心地把试卷交给牛开复师兄，师兄略略浏览，说道：“嗯，整体上还可以，不过有些地方就有些不够完美了，比如这道题：&和&&的区别是什么？”

“&是位运算符，表示按位与运算；&&是逻辑运算符，表示逻辑与。没错啊！”

“再想想，仅有这些吗？”

“不是吗？……哦，对了，&也可以作为逻辑运算符，表示逻辑与。”

“其他的呢？&在表示逻辑运算与的时候，和&&是一样的吗？”

“哦！&&是短路运算符，而&不是！”

“所以嘛，想问题一定要周全。Java 笔试中会有很多像这样的题目，要求你区别性质相似容易混淆或者仅仅名称相似含义相差甚远的两个概念。类似的题目比如：

- 区别 Collection 和 Collections
- final、finally、finalize 的区别
- sleep()和 wait()有什么区别？”

“第一个我知道，Collection 是集合框架类的上层接口，Set 和 List 均继承自它；而 Collections 类是针对集合框架类的一个工具类，它提供一系列的静态方法对各种集合框架进行搜索、排序、线程安全化等操作。”

“嗯，不错。光说得流利是不够的。运用的时候别忘记了。剩下的那几道呢？”

“剩下的这几道题用的时候能分辨出来，但是要说，还真有些无语呢。”

“还是领悟得不够吧，没关系，冰冻三尺非一日之寒，慢慢练习慢慢体会就行。”

**提示** 笔试中的选择题和简答题大概就是考查这些方面的内容，也还会有 Java EE 或 Java ME 以及数据库方面的知识。对于选择题还好处理些，主要是简答题对于应聘者的思维能力和表述能力有比较高的要求，读者应该在实践中多体会。

牛开复师兄继续说：“下面我们来看看你的编程题做得怎么样。”

题目：已知一字符串，长度不定，要求找出在字符串中出现次数最多的字符，并输出其出现次数。

“师兄，这个题目我的做法是先把字符串转换成数组，然后遍历这个数组，对于每一个字符，分别向前和向后查找有无相同的字符，然后记录下出现的次数，并与已有的最大值 max 比较，若比已有最大值 max 大则将此次出现次数赋值给最大值 max。”

“嗯，这么想没错，但是你觉得考虑欠妥吗？如果某两个字符的出现次数相同且都为最大值呢？岂不是会漏掉？”



“对对，这个没有想周全。那我应该在后台维护一个整形数组，长度和字符串一样长。然后，与一开始的方法一样遍历数组，整形数组记录相应索引的字符出现的次数，同时得到最大值 max。然后再次遍历字符数组，只要它的出现次数等于最大值 max，就将其输出。”

“不错，这道题这么答可以。这算是大众解法。你再想一想还有什么方法吗？”

“别的嘛，我想想……想不到了，还请师兄指点一下啊！”

“我说的这个方法不算是最佳解决方案，但是却提供了一种新思路。那就是哈希表的方法。”

“哈希表？这里也可以用哈希表？”

“没错，我们把字符串生成的字符数组中每个元素当作要存放的值。新建一个字符数组为哈希表，长度为字符串的长度，这样能保证存下整个字符串。随便定义一个哈希函数，比如  $h(x)=x\%length$ ，x 为字符的 Unicode 值，length 为哈希表的长度。”

“哦，我有点理解师兄你的意思了。然后是不是进行存储？把字符数组中的每一个字符都通过哈希函数计算出地址，如果地址冲突，就判断待存字符是否与已存在字符相同，不相同则根据处理冲突方法继续寻找散列地址，相同则计数器加 1。”

“嗯，最后和第一种方法一样，先遍历出最大值 max 再把所有与最大值 max 相等的字符输出即可。怎么样，这种方法是不是让你眼前一亮啊？”

“的确是这样，能想到这种方法，肯定对算法有很深的体会。”

**专家分析：**编程题一般分为两种。

- 第一种主要考查应聘者对 Java 语言的驾驭能力，比如会考多线程编程、内部类的实现、继承多态等 OO 思想的特性等。这种题目需要较多的编程实践和对细节的把握。
- 第二种就像故事中的那道编程题一样，考查的是应聘者的逻辑思维能力和对算法的理解，希望各位同学在下工夫啃 Java 的时候，能分出精力来研究数据结构和算法方面的知识，那是不以任何语言 and 平台为转移的。

牛开复把卷子翻到最后，说道：“编程题我们就谈到这，最后我们来谈谈有可能出现的智力题，这种题目还是很锻炼思维的。”

题目：

- 有一辆火车以每小时 15 公里的速度离开洛杉矶直奔纽约，另一辆火车以每小时 20 公里的速度从纽约开往洛杉矶。如果有一只鸟以每小时 30 公里的速度和两辆火车同时启动。从洛杉矶出发碰到另外一辆车后返回，依次在两辆火车来回飞行直到两辆火车相遇。请问这只小鸟飞行了多长距离？
- 想象你在镜子前，请问为什么镜子中的影像可以颠倒左右却不能颠倒上下？
- 如果你有无穷多的水、一个 3 升的和 5 升的桶，你如何准确称出 4 升的水？

“师兄啊，我看到第一题就有些茫然了，这要算很久的吧？剩下两个我也没太想明白。”

“你是研究书本多了，思维变得有些僵化了，先看第一道题，如果换个角度考虑，小鸟飞行距离的另一种算法是用速度乘以时间，速度是已知的，而时间就是两辆车相遇所用的时间。于是这道题就变得非常简单了吧？”

“哦。那第二道题也应该是换角度考虑的吧？”

“没错，第二道题你可能会想到物理上的知识、平面镜成像啊等，不过这道题的正确答案是：如果把镜子放在天花板或地板上，镜子中的影像就会是上下颠倒的了。”

“第三道题呢？”

“第三道题很经典。答案是这样的：把5升的桶装满水，再倒入3升的桶直至其满溢，然后清空3升的桶，再把5升桶中剩下的水（2升）倒入3升的桶中，最后往5升的桶中装满水，倒入3升桶中直至其满溢，5升桶中剩下的水就是4升了。”


“哇，看来身为开发人员思维要活跃啊。”

“那是必需的啊，呵呵。蔡佳娃，经过今天的小试牛刀，感觉怎么样啊？”

“笔试还真不是个可以偷懒的活，不光要看书，更重要的是培养思维啊。”

“呵呵，你脑子还可以，肯定没问题的。我等你的好消息，好好考呀。”

“嗯，师兄，我一定全力以赴！”

 **提示** 作为一个开发人员，永远做的是思维的运动，在此建议各位读者把工夫下在平时，临时抱佛脚或许会有用，但厚积薄发来得更为爽快。

### 4.2.3 笔试小结

虽然历来考试高一尺，作弊高一丈，但是笔试这一关的蒙混几率却不是很高，反而会有一些“高手”因为平时对基础知识和一些小问题不太留意而 fail 掉笔试，与心目中的公司失之交臂。因此，作为初学者不可毛躁，应该在每次实践中注意细节，勤于积累。

## 4.3 面试——最难的 BOSS

有句俗语说得好：“是骡子是马拉出来遛遛”，闯过了简历关和笔试关，面试这一关就是拉出来溜溜的时候了。前面两关或许还可以躲躲闪闪、坑蒙拐骗，到这里可就是丑媳妇见公婆了。面对这个游戏中最难对付的 BOSS，该如何将其击败呢？

### 4.3.1 面试面什么

话说上次蔡佳娃在潜心向牛开复师兄请教笔试技巧之后，自己又很不放心地狂看了两天书，然后就杀气腾腾地去笔试了。后来他在跟师兄的电话里用“虽艰难，终涉险”来形容这次面试。现在，摆在蔡佳娃面前的就是这道面试关了。

“师兄啊，这个简历和笔试，或多或少我们在学校里也大概接触过，比如写报告和考试，可是这个面试我们可真是少有经历啊。后天我就得去面试了，师兄你给我支支招吧。”

“找工作这回事，越往后考查得越全面，越不大可能靠临时准备应付过关。面试主要看你有没有技术和非技术上的问题，是不是适合目标公司。”

“我听说面试也要看考官是谁，HR 和技术不一样吧？”

“嗯，HR 考官主要看你这个人怎么样，比如你的谈吐举止、反应能力和思维能力、有没有进取心、有没有工作热情等。而技术考官则要看你的专业素质，比如会问一些技术上的问题、会针对你的简历提出相关的问题以检验简历的真实度等。”

不同公司 HR（人力资源）和技术考官的面试顺序一般有很大的不同。小一些的公司人力资源方面的因素在最终决定时占的比例相对较少，而大公司还是很重视人力资源方面因素的。

不过一般情况下还是技术考官，即应聘者将来的上级发表主要意见，如果技术面试通过，HR 的面试就是大致看一下应聘者有没有大的毛病。不过有时 HR 考官同时也是技术考官，也有些时候会只有技术考官的面试。

“师兄，那 HR 面试的时候具体会问些什么呢？”

“一般的 HR 都不是搞技术的，所以问的问题不会很专业，但是可不要小觑 HR 的本事，往往专业技能很全面的应聘者却会被 HR 看似犀利的问题逼得结结巴巴。”

“啊，我可不想被 HR 给整崩溃，师兄你说说他们会怎么提问我们啊？”

“最一般的情况他/她会让你做个自我介绍，或者讲讲自己的特长和不足，或者谈谈自己的职业规划，谈谈如果顺利进入他们公司你有什么打算，如何开展自己的工作，自己对薪水的问题是什么样的看法等。”

“天哪，这些问题还真的从来都没有想过呢。要不是听师兄你这么一说，只怕到时候被问到，也憋个大红脸啥也说不出来说，那样就惨了。”

“呵呵，别这么不自信。其实 HR 考官在问你这些问题的时候，一般情况下并不是真正想要听你的答案是否正确，而更多的是分析你的回答思路是不是清晰、口齿是不是伶俐、有没有条理、有没有过分的紧张和不安。有时候还会问一些很突兀的问题呢。”

“啊，这么夸张啊！”

“是啊，你要想想，身为 HR 面试考官，他们也需要创新和进步啊，所以不能总停留在那些中规中矩的问题上吧。比如他们会问你我们凭什么录用你？怎样对你你才会辞职？上级不喜欢你怎么办？你的家人不支持你的工作怎么办？想在公司混到什么职位？”

**专家分析：**尽管 HR 考官大都并不擅长技术，但是就算你技术再过硬，往往面对 HR 考官丢过来的问题也会有些接招不暇。不过，醉翁之意不在酒，在貌似与技术能力无关的问题背后，聪明的考官往往看中的是以下几点：

- 举手投足：从求职者进门开始，考官就在观察，每一个小动作都会反映求职者的性格特征，都会成为考官为其打分的依据。
- 交际口才：考官通过求职者回答问题的方式和内容，会对其表达能力和交际能力有一定的了解。这些能力虽然在技术上帮不上忙，但是须知 IT 行业可不是扛着技术大旗就可以打天下的。
- 工作态度：考官会从求职者的言语和举止中看出其工作态度是否认真、热不热情、积不积极、稳不稳重。
- 应变能力：考官会提问一些比较突兀的问题，通过观察求职者的回答延迟、回答流利与否、答案是否条理分明，来判断求职者的应变能力。

“牛师兄，我觉得 HR 考官有时候挺像记者的，提的问题很不好回答。”

“呵呵，HR 考官就是靠这些貌似很不好回答的提问来为自己的公司挑选合格人才的。真正的人才，不应该只是技术能力强，为人处事、交际、应变能力都是同等重要的。所以啊，不要小瞧

HR 考官，他们往往会从你手忙脚乱的表现中看出你的破绽。”

“让你这么一说，HR 面试这关也是很险咯？”

“HR 面试还不算最困难，因为其主要是为技术考官的面试提供一些参考意见，往往还是技术考官说了算呢。”

“那技术考官面试的时候会怎样出招呢？”

“技术考官一般没太多心思研究你这个人的性格特征，他们直接上来就问技术问题。所以到了那时候你也就别谦让了，有多大能耐是多大能耐，尽最大努力促使考官点头吧！”

在这里需要提醒一下，笔试中曾考过的题目，在技术考官面试这一关，是完全有可能会再次问到的。但并不是说，技术考官的面试就会稍微简单一些。如果笔试中某道题答得不错的话，在这一关同一道题的回答却可能让技术考官眉头紧皱。

即使是相同的题目，却是一说一写不同的答法，技术考官可比 HR 考官敏锐得多，你回答问题的思路清不清晰、对这部分知识掌握得透不透彻，他们马上就可以看出来。因为写在纸上的知识和嘴里说出来的知识绝对是不同的。

“嗯，看来还是不能掉以轻心啊。”

“是啊，技术考官除了会问你这些问题，还会根据你的简历让你介绍一下项目经验的细节呢。比如技术考官会拿着你的简历问这些问题：

××环节具体是怎么实现的？具体采用了什么技术？

××项目过程中出过什么问题？怎么解决的？

在××环节有没有试着用这种方法来解决？为什么不用？

……”

“哇，师兄，我有点害怕了耶。”

“不用害怕，能够挺到面试，如果没有弄虚作假，都不会是等闲之辈，拿出自己的本事，只要相信自己能行，就不怕他们问。”

**专家分析：**技术面试是比较有决定性的环节，主要考查的是求职者的以下两个方面。

- 技术能力

项目是不是真做过，是不是真明白，是不是真掌握了，特长是不是真精通，是不是蒙混过关的，一问便知。所以这里也是对简历的一个检验。

- 思维能力

求职者对于技术考官提出问题的回答，便是其脑中思维的外在体现。任何一家用人单位都会想录用那些逻辑思考能力强、思维活跃的人才。

### 4.3.2 支招面试

前面一小节本书介绍了面试这一关中 HR 考官和技术考官所考查的人才的不同方面，本小节将继续跟随蔡佳娃的脚步备战面试。

“师兄，听你这么一说，面试，尤其是技术考官的面试，的确是来选拔人才的。看来这一关的确不是能混过去的啊！”

“那是当然，这算是求职过程中至关重要的一关，一面定江山呢，所以千万不可马虎。当然也不用过分要求自己。毕竟学问有高低，专业的思维能力也不是一天两天能够练成的。过分地准备或许会弄巧成拙，不过也不可打无准备之仗。”

“那师兄你给我指点一下迷津吧，不管怎么说我也得试试看。”

“嗯，很好。首先看 HR 面试。其实与 HR 面试也可以算得上是聊天。HR 考官只是想了解你，看看你是不是适合人家的公司。所以你首先就是要做到不紧张，轻松一些。”

“哎，这一点我到了现场再慢慢安慰自己吧。”

面试的准备工作基本上该做的都应该在平时做完，比如技能水平提升、交际能力与口才的锻炼、个人性格态度的锤炼等。到面试那一天，只要把自己的形象好好整一整即可。

模样我们没办法优化，而着装和气质是可以后天打造出来的。不管结果怎样，都要面带微笑，以一种阳光、热情、诚实、有进取心的形象示人。拿出买卖不成仁义在的气势来，不要对自己定太高的目标，这样对 HR 的面试也是十分有利的。

“那具体一点说呢？比如 HR 考官来面试你了，该怎么应对啊？”

“呃，其实有些时候呢，HR 考官也像是在拷问囚犯，想方设法下圈套骗你说出自己的缺点，这时候就要注意，不可全部老实地作答提出的问题。很多问题都是有圈套的。比如考官问你有什么劣势，或者问你编程过程中经历过的最大的困难等。”

“我就撒谎，说没有？”

“不是，这个可以有。只是你不能太老实。应该巧妙地避开对你应聘工作不利的因素。让考官知道虽然你有一些缺点，也犯过一些错误，经历过失败，但还是不影响目标公司的岗位要求的。因此，适当地掩盖还是必要的。”

**提示** 考官在问问题时，都要注意到不仅自己的思维能力在被考验，自己的表达能力也在被考验，所以尽量不要让自己的嘴巴误了事。回答问题要干脆、斩钉截铁，不要吞吞吐吐，少说“呃”、“嗯”、“这个”、“那个”等。

“嗯，我明白了，还是不要过分牵强附会，有一说一是吧？”

“那是当然啊，人家 HR 哪个不是面过 N 张脸的人啊，你扯得太荒唐他们怎么会不晓得啊。尤其你再连说话都不连贯，就算你是真品也怕被人家鉴定成赝品呢。”

“哈哈，师兄说得很对，说话流利也是一种干练的表现。”

“是的，语言是思想的外在体现嘛！”

“那么，师兄啊，哪些地方我们还可以稍微做做假啊？”

“如果说还有的话，就是在 HR 考官研究你的时候，你也尽量分析分析考官的心理，尽量投其所好，在不恶性欺骗的基础上尽量让考官喜欢上你。比如考官是个女的，又问你关于家庭的问题，你就回答说自己会尽最大努力协调好工作与家庭的关系等。”

“如果考官是个明显的工作狂，我就说我一切为了工作，工作第一。家人一定会理解我、支持我的，对不对？”

“呵呵，学得蛮快的嘛，差不多就是这个样子。”

**专家分析：**面对 HR 考官的时候，本书给求职者的对策就是尽最大可能让考官喜欢上你，具



体需要注意的地方如下所示。

- 外在形象，海水是真的不可斗量，但人经常被貌相。所以注意自己的着装谈吐，让自己的合格形象打响面试求职的第一炮。
- 人格魅力，虽然谈魅力遥远了些，不过还是要尽可能让自己显得口才好、讲礼貌、有进取心、真诚、热情……当然如果这些特质读者并不具备，也不用过分伪装。
- 适当提升，毕竟 HR 考官不是技术考官，所以在技术层面上的问题，还是可以夸张一些的。其他问题也要看情况而定，最好不要老实巴交地问什么答什么，说的话要为自己的目的服务。
- 做到无懈可击，面对 HR 考官丢过来的犀利问题，要学会避重就轻，避实就虚，尽量不让她看出自己的破绽。
- 提前准备答案，对于一些提问几率很大的问题如自我介绍、优势劣势分析、如何开展自己的工作、为何辞去前一个工作等可以提前准备好。记住一定要尽量从用人单位的角度来考虑和准备问题答案，这样会使求职者在面对考官时更加成竹在胸。

“师兄你再说说遇到技术考官时应该怎么对付啊！”

“面对技术考官你就没必要装了，把自己所知道的、掌握的、精通的全都抖出来。还有啊，如果你的简历写得很高深以至飘渺，那就要小心了。”

“啊，难道考官还会拿着简历对口供吗？”

“是呀，要知道这可是一锤定音的环节，考官肯定要保证招聘进的人是真的可以做事的有用人才，养闲人可不是 IT 行业的习惯。”

“那我该怎么办啊？”

“这样吧，你回去准备一下，我们明天来一个模拟面试吧，只是技术这一块的，要不说了也是纸上谈兵。”


“嗯，也行，像我这样的菜鸟最喜欢的就是举例子、打比方。呵呵。”

“你回去要做的就是：提前准备。自己做过的项目要提前准备好技术讲解和项目架构图，到时候我问哪里你就答哪里。”

“项目架构图，那是什么啊？”

“就是项目功能实现的结构图示，你可以去网上搜一搜，试着画一下，明天我们模拟的时候我再帮你看看。”

“嗯，明天见考官！”

 **提示** 技术考官往往看人很准，尤其是看新人，所以在这一环节最好还是实事求是，好好准备一两个拿得出手的项目经验，这里毕竟不是学校。

### 4.3.3 面试演习

实践是检验真理的唯一标准，纸上谈了这么多兵法，也该拿到战场上运用一下了。提前感受一下兵书上的轻描淡写和战场上真实残酷的不同，或许对于一个新兵的成长是最有帮助的了。

“蔡佳娃，你准备好了吗？我们要开始技术考官的面试了。”

“师兄，不要手下留情啊，放马过来吧！”

演习开始！

“你好，请自我介绍一下。”

“啊，不是说是技术考官吗？怎么会问 HR 的问题啊，师兄？”

“你好，能自我介绍一下吗？”

“哦…呃…你好，我叫蔡佳娃，是××理工大学计算机专业大四的学生。我在大学期间……通过大学期间的学习和参加校外的培训，逐渐掌握了 Java Web 开发的一些知识……与技术，并且也参与了一些项目的工作，积累了一些经验。我对贵公司有一定的了解，结合自己的能力和经验，我觉得自己还是可以为贵公司尽一份力的。”

这是蔡佳娃这样的新人最容易暴露出的一个问题：想当然。虽然说了是技术面试，但是也不能想当然地将人力资源面试和技术面试完全区分开，制定本来就不存在的虚拟规则。同时，对于突兀问题回答得仓促结巴，也反映出应变能力差的一面。

读者要注意，用人单位在招聘技术人员时是十分重视应变能力的。因为世界上没有任何两个项目是完全相同的，作为一个合格的开发人员应变能力十分重要，读者朋友们平时要注意尽可能多地培养自己这方面的能力。

“嗯，现在 IT 公司这么多，你为什么选择我们公司？”

“那个……贵公司对行业形势判断很好，而且我认为以自己的能力，是可以胜任这项工作的。另外，我选择贵公司也是因为这样离我的家近一些。”

“那么，你觉得如果你被录用，摆在你面前的最大困难将会是什么？”

“我觉得可能是工作环境的问题吧，毕竟在学校和公司的工作环境都不一样，可能要花些时间来适应吧。”


“你觉得你哪方面最不行？”

“我觉得我有时候太钻牛角尖，不达目的不罢休，可能会耽误一些正事的进度。”

这一部分蔡佳娃表现得很不错，除了那句“离我的家近一些”需要被列为毫无意义的废话，因为这句话跟面试无关而且也会让面试考官觉得此人不够稳重和严谨。而关于自己不足的回答就很精彩了，既没有正面否定不谈，又讲了一些既像优点又像缺点的不足之处。

“呃，你的简历上面说你对 Java Web 开发掌握得不错，讲一下 Servlet 的生命周期吧。”

“嗯，Servlet 是在 Web 服务器启动的时候 new 出来的，然后如果有 http 请求到来时，就会调用自己的 service() 方法，根据请求的不同执行 doGet() 或 doPost() 方法。Servlet 是可以同时处理多个请求的，当服务器停止的时候，Servlet 也会被销毁。”

 **提示** 关于 Servlet 生命周期这个问题，蔡佳娃回答得也不错，在谈到技术的时候，最好少用口头语，尽量用技术用语，而且语言越精练越好。

“好的，说说你做过的项目吧。”

“我在大三的时候参与过一个移动环境监测系统项目的开发。”

“是吗？请讲一讲项目的总体功能吧。”

“好，这个项目主要是实现对环境监测点的环境质量进行上报和查询，监测的内容包括水质、大气和违法排污现象。分为手机端和 Web 服务端。”

“具体手机端和 Web 服务端的功能是怎样的？”

“手机端主要负责大气和水质情况的上报和查询，Web 服务端具有手机端功能的同时，还作为服务器具有接受上报数据、发回查询数据、管理数据库等功能。”

（递过去一张纸）“请你画一下项目架构图吧。”

“啊，不用，我带着呢。（见图 4-1）”

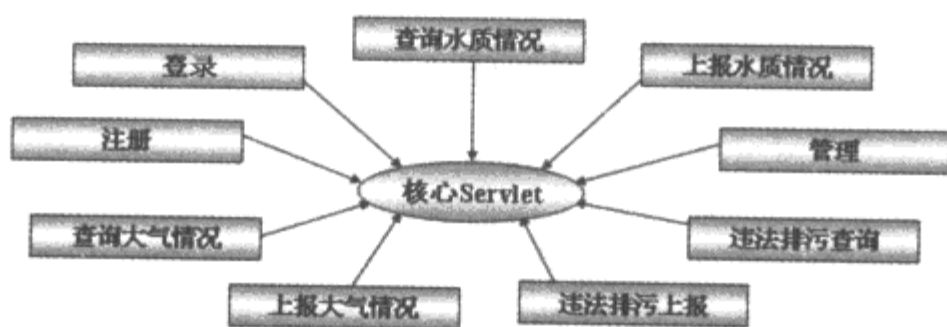


图 4-1 蔡佳娃的项目架构图

这里又暴露出一个问题，首先不说蔡佳娃画的这个项目架构图是好是坏，他不应该将一张已经画好的项目架构图递过去，因为在这种情况下面试官是希望蔡佳娃自己现场画的，这样还能考查一下这个人的知识再现能力。

“啊？你这个图……，好，你详细说一下你在这个项目中都做了些什么吧。”

“我主要负责 Web 服务端的设计和开发，这个项目中要有两个 Servlet，一个负责处理手机端的请求，一个负责处理 PC 浏览器端的请求。”

“那服务端是如何接收手机端请求的呢？”

“首先在 Servlet 的 `doPost()` 函数中调用 `request.getInputStream()` 函数得到请求的输入流，并将其封装到 `DataInputStream` 对象中，再调用 `DataInputStream` 类的 `readUTF()` 函数获得字符串格式的请求数据。”

“服务端是如何响应手机端请求的呢？”

“Servlet 通过调用 `response.getOutputStream()` 函数得到输出流，并将其封装到 `DataOutputStream` 对象中，再调用 `DataOutputStream` 的 `writeUTF()` 函数发送响应数据。”

“开发的过程中有没有出现过什么值得留意的地方呢？”

“有，因为要面向手机端，而不同的手机平台对 Java ME 的支持有细微的差别，比如当面向的手机终端为诺基亚时，要在每次向手机端发送数据时在 `writeUTF()` 函数后面调用 `flush()` 函数，否则手机端就不会收到响应数据的。”

**提示** 对于项目中值得留意或收获最大的地方这种问题，如果实在没有就不要乱说，因为技术考官都是上过战场的老兵，不恰当地班门弄斧，最有可能给自己画蛇添足。

“你觉得这个项目的缺陷在哪里？”

“我觉得界面还是不够友好，全是数据啊、表格啊，客户可能看了不会太喜欢。”

“如果要你再优化一下这个项目，你会怎么做？”

“我想在 Web 端用 AJAX 技术做一个环境质量监测图，这样视觉效果会好很多，同时手机端也可以用 GameCanvas 实现类似的视觉效果。”

“那么，谢谢你的作答，我们会给你电话的。”

“好的，谢谢你，再见！”

演习结束！

总体来讲，蔡佳娃的这次面试初体验还是比较成功的。除了开始的几个貌似应该归到 HR 面试题中的问题，蔡佳娃在技术层面上的回答很简练准确，这种比较干脆的表达方式是比较受技术考官欢迎的，看来准备充分些的确是会有些优势的。

“师兄，怎么样？我表现得还行吗？”

“我先不说，你觉得自己表现怎么样？”

“我觉得前面那几个问题回答得不是很好啊。”

“那不是一般地不好啊，我昨天跟你说的避免犯的错你基本上都犯了呢。”

“啊，不会这么夸张吧。不过，师兄你说话不算话嘛，明明说了是技术面试，怎么还会有自我介绍之类的问题啊。”

“你们就是在学校待得太久，总想把所有事都摸索出一个固定的规律来，然后再按照规律制定一个固定的策略。求职游戏可不像八股文那样有这么多条条框框，它是很自由的。所以如果你只是按照固定的模式去准备，而不想着如何应对突发情况，那就有风险了。”

“对，现在已经不能像在学校考试那样押题或是猜题了。”

“就是嘛，你看你的回答，又是口齿不清，又是跑题，问你为什么选择人家公司你居然还会说因为离家近。”

“哦……我那时有点懵。”

“还有，问你可能最大的困难是什么，虽然那个困难不算什么，但是你说话的语气却让考官我很担忧啊，人家还以为你在担心自己没办法适应新环境呢。”

“哎，看来要注意的问题还真不少啊。那其他的不足呢？”

“其他问题回答得都还可以。下面我们看看你的项目问题。你这个项目还算不错，不过你看看你的项目架构图，画的叫什么啊。要是没有 Servlet 这个词，人家都不一定会认为是个软件项目的架构图呢。我把我当年找工作时用到的项目的架构图给你看看。（见图 4-2）”

“这是一个学生信息管理系统的项目架构图，写得比你的要详细多了吧。”

“不过，除了比我画得多以外，其他也没有什么区别吧？”

“区别可大了！首先你的图画得太不具体，只是把功能模块列出来了，那根本不是架构图，只是功能罗列图。架构首先是你的项目的详细物理结构，所以应该是由一个个项目文件组成，每个项目文件会执行一个或多个功能，数据流向就说明了这些功能的执行结果和去向，而且也没有数据库模型。”

“真的是好复杂啊，当时师兄你找工作的时候就是带着这个图去面试的吗？”

“怎么会带着呢。你想得太好了。都是给一张纸，现场画的，所以我刚才看到你居然拿出一张图来就很诧异了。”

“啊，不是吧，这么复杂怎么能画得全呢？”

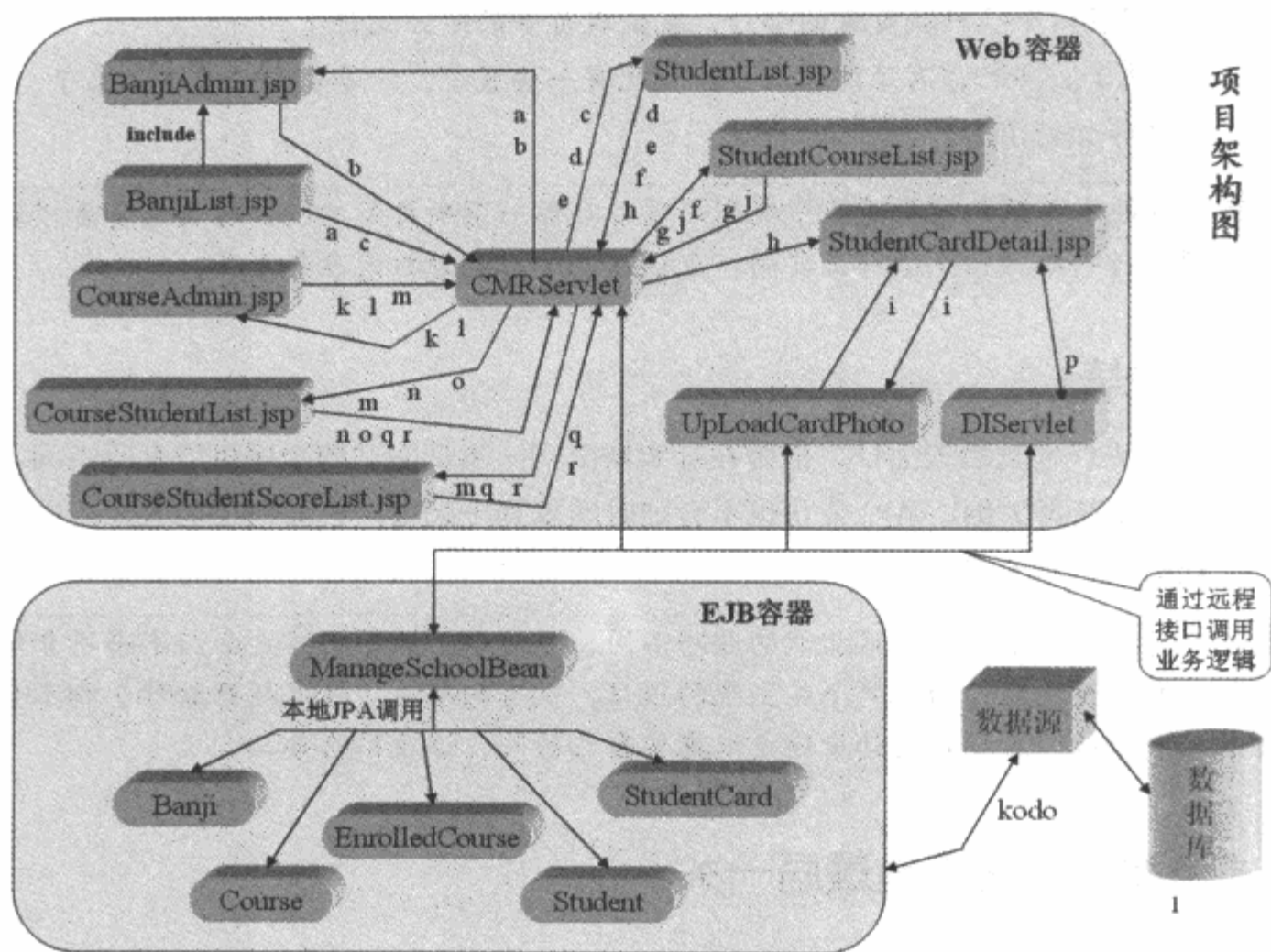


图 4-2 学生信息管理系统架构图一

“这个图给你看是仅供参考的，真实情况下并不需要全部画出来的。你不要把顺序搞错了，项目架构图是项目开始之前画出来的，之后的项目开发都以此为据。而不是把项目写完了之后总结出来的。”

“啊，幸亏有师兄你提醒啊，我差点就搞错了呢。还有啊，师兄你这个项目架构图上的 abcd 等字母是干什么用的啊？”

“这个是动作编号，是执行的具体功能。这是对照图。（见图 4-3）”

动作编号对照表

动作编号	动作内容	动作编号	动作内容
a	删除班级	i	替换照片
b	添加班级	j	删除学生选课
c	查看学生明细	k	添加课程
d	添加学生	l	删除课程
e	删除学生	m	查看选课学生
f	查看所选课程	n	添加选课学生
g	从学生删除选课	o	删除选课学生
h	查看学生证	p	获取照片
q	提交成绩	r	暂存成绩

图 4-3 学生信息管理系统架构图二


“哦，那当时师兄简要地把这个项目的架构图画出来后，考官是怎么问你的啊？”



“当时考官问我这个项目我负责的部分，我说我负责的地方就行了。”


“哦，我记住了。师兄，有了这次演习，我就有一些底了，就等明天一面定江山了。呵呵。”

“OK，师兄等你好消息！”

 **提示** 在技术面试之前把简历中提到的项目准备一下项目架构图，对自己负责模块的技术细节问题进行一些回顾是很有必要的，免得在实际面试时打无准备之仗。


#### 4.3.4 面试小结

本书虽然是先谈笔试后谈面试，但是在真实情况下，笔试面试的顺序可能有所不同。而且笔试和面试没有一个统考大纲，各个公司也不会抽时间聚到一起来研究如何招聘人才。因此，本书介绍的是大众情况下的求职之路。

 **提示** 有些情况下不仅笔试面试的顺序不同，笔试和面试的内容也会与本书所介绍的有所不同。而且，说到底，面试是个纯主观的项目，所有的题目（技术题目除外）都不会有唯一标准的答案。所以希望广大读者还是以充实自己第一，谋攻策略第二。

### 4.4 试用期——这才是最后一关

如果各位读者认为自己已经被“拉出去溜了溜”，而且顺利打败了游戏中最难的 BOSS，以为高枕无忧那就错了，试用期才是这个游戏中的最后一关。如果前面的几个关卡还主要注重的是竞争和考试，那这一关就是主要看实践了，实践是检验真理的唯一标准。

 **提示** 试用期这一关并不是最难的，但却是最重要的，每一位求职者都不想功败垂成，所以，平安度过或者靠自己能力缩短试用期，就是步入 IT 这个江湖的最后一跃。

#### 4.4.1 试用期考查什么

每一位进入试用期的求职者都是踌躇满志、血气方刚的。但是，经过了前面几关的洗礼，到了这里仍然不可以放松心态。试用期时一只脚已经踏入了 IT 这个行业，所以试用期所做的是为了外面的那只脚。

蔡佳娃到了面试那天居然人品爆发，面试结束后立刻就接到电话宣布进入试用期。蔡佳娃当然不忘感谢一直帮助他的牛开复师兄，马上打电话给师兄要请他吃饭。当然，在饭桌上，蔡佳娃也不免要向师兄咨询一下传说中的试用期。

“师兄，你说试用期是个啥玩意啊？”

“首先呢，试用期可长可短，完全取决于你在公司的表现。其次，试用期可以升级为就业合同，也可以走向下次求职。”

“那师兄照你这么说，试用期也不是那么保险的呗。”

“看看，你又想放松了是吧。没有什么事情是万无一失的。况且 IT 这个行业又是个技术更新快、瞬息万变的行业。”

“是是，师兄说得对。那试用期公司考查我什么呀？”

用人单位判断是否让一个试用期的人转正的条件有如下几个，这也是用人单位在试用期重点考查的方面：

- 第一个依据是此人是否名副其实，像他简历所说的那样，该精通的知识都精通，是否有弄虚作假的嫌疑。所以如果前面几关水分大，到了这里就有可能露出狐狸尾巴。
- 第二个依据是此人是否为公司创造了价值。公司存在的唯一目的是盈利，说别的都是套话。所以如果一个人没有为公司创造价值，那这个人就可能要为公司节约一些财富，即被辞退。现实很残酷，所以在试用期要拿出百分之二百的努力来拼。
- 第三个依据就是此人的人品和态度怎么样，对工作是不是负责，是不是有工作热情，是否诚实可靠，跟上级和同事关系怎么样等。
- 第四个依据就是此人是否具有一定的不可替代性，当然这里的不可替代性并不是指没了某个人公司就不能运转。而是根据你現在为公司创造的价值，公司得计算着辞退你再另聘他人和留着你继续培养哪个更合算。如果是后者，那么你就具有一定的不可替代性。

“师兄，这几个条件虽然听起来很残酷，但是道理却真是这样啊。”

“是啊，IT行业可不是宫廷乐队，所以休想‘滥竽充数’。”

“师兄你说得没错，我在试用期内一定全力以赴，尽快转正成为一名真正的开发人员！”

“嗯，全靠你自己了，放心大胆地使唤自己吧！”

**提示** 考验无处不在，没有永远安逸的地方，“生与忧患，死于安乐”。既然走到了这一关，就要继续坚持下去，功败垂成是最遗憾的。就像故事里说的那样，公司存在就是为了盈利，是否能为公司创造价值，就是衡量各位求职者能否转正的关键。

#### 4.4.2 多做什么，少做什么

既然试用期也是如此“险恶”，蔡佳娃不禁心里发怵。他十分不想自己已经迈过去一只脚还要被踢出来，于是吃过饭后还是缠着牛开复师兄不放。

“师兄啊，明天就是我试用期的第一天上班了。你不给我点意见吗？”

“让自己安然度过试用期很简单，想想自己从前上小学是如何积极表现争取拿小红花的，原理大概差不多。”

“啊，师兄你太会开玩笑。”

“呵呵，在试用期内，你的底线是要保证积极按时完成上级交给的任务，如果这个都做不到，那真的与小红花无缘了。”

“嗯，那个的确是根本。其他方面呢？”

“试用期内你应该注意多做些什么，少做些什么，让自己尽快成长为一个开发人员。”

在试用期内，要多看、多听、多做。从学生变成开发人员，要学的东西很多，而一般也不会有人专门为你搞个培训，所以只好自己偷着学，通过观察其他人，慢慢找到开展工作的门路。剩下的就是靠自己的专业技能为公司出力了。

同时，要少说空话、废话、无用的话，少唱高调，要明白自己已经不是一名大学生了，少把自己大学时逃课等坏习惯带到公司来。一个新人，入行的形象最好是低调、踏实、肯干，所以应

该尽量避免让自己哗众取宠。

“对对对，是该低调一些，何况我还是个菜鸟。”

“其他方面，你在公司应该不怕脏、苦、累。多干活，少埋怨。甭管分配的工作喜不喜欢干，先拿下再说。展现你的工作热情和旺盛的战斗力和战斗力，这可是一些老员工所没有的。而且我刚才说的低调可不是有新创意憋着不说，虽然我们是新手，可是该出手时就出手，风风火火闯九州嘛。”

“呵呵，师兄，你说得很正确嘛。我会好好做个菜鸟的，当然不会做太久，我要立志做个像师兄这样的高手！”

“哈哈，努力就一定可以做到。师兄我也没有那么厉害啦。我们共同进步！”

#### 4.4.3 试用期小结

试用期是求职的最后一步，也可以说是真正上班的预备期。虽然还不是正式员工，但是也要拿出比正式员工还高的热情来，主动出击而不是坐以待毙。既要初生牛犊不怕虎，也要小心驶得万年船。最重要的还是用自己的专业技能为自己赢下一份满意的正式用工合同。

### 4.5 本章小结

本章向读者介绍了求职过程中所经历的各种关卡和具体攻略。希望能对广大读者的求职之旅有所帮助，求职是每个人步入职场的第一步，摆正心态，认清自我，确定目标，不骄不馁，昂首前进是本书对广大读者的最好建议。



#### 提示

尽管本书提出了种种应对策略，但是各位读者要明白，蒙混过关不可取，不然就会让自己总是徘徊在躲初一和躲十五之间。真才实学才是硬道理，做最好的自己就是成功。

# 第 5 章 步入江湖——做事的学问

闯过了简历、笔试、面试、试用期等重重关卡，展现在各位读者面前的就是波澜壮阔的 IT 江湖了。进入职场，面对的将是另一番天地。如何做人、如何做事，都是一门不小的学问。在 IT 职场做事，除了技术水平的高低需要重视之外，很多其他学问也是必不可少的。本章将向读者介绍一些在职场做事的学问，希望对各位职场新手有所帮助。

说到步入江湖，谁也没有蔡佳娃同学兴奋，因为他刚刚结束了试用期，已经转正成为一名 Java 开发人员了！不过初出茅庐、经验不足的他面对这个全新的环境，还是有些应付不来的。幸好我们还有高人师兄牛开复为他保驾护航。

## 5.1 身为菜鸟

步入职场，第一个头衔只能是职场菜鸟，做菜鸟的时间长短因人而异，但是每个人都要经历这个阶段。身为一个菜鸟，面对工作，应该是什么样的心态和做法呢？应该争取什么，避免什么？本节将要探讨的，就是如何做一名合格的菜鸟。

### 5.1.1 打碎牙齿往肚里咽

或许一开始就谈打碎牙齿这么血腥的问题不好，但是职场不是过家家，而是擂台。在弱肉强食的竞争之下，IT 菜鸟必须学会这一招。否则如果打碎牙齿吐出来的话，那么吐出来的将不仅是牙齿，还有自己失败认输的白旗。

当然了，这里的打碎牙齿并不是真的每天去公司挨揍。打碎牙齿往肚里咽是指菜鸟在初期面对工作的一种态度，核心思想就是不示弱、不露怯、不言苦、不说累。这么说或许会有些过于夸大其词，不过字字真言。

终于成为一名正式的开发人员，蔡佳娃为此欣慰了很久。毕竟这么多年的学校教育和自己的努力都没有白费，终于是踏进了这个门槛，离自己的榜样——牛开复师兄——又近了一步呢。

万事开头难，蔡佳娃不是天才，肯定也会碰到菜鸟都会茫然的问题，出于剧情需要，蔡佳娃是必须要出现问题的，而且是各种各样的问题。

“师兄，最近忙吗？我最近好痛苦啊！”

“哦？你不是刚刚转正吗？正是激流勇进的时候呢，怎么又痛苦了啊？”

“哎，转正是转正了，工资也上去了。可是我发现最近上级给我分配的任务是越来越使我不得开心颜了。”

“小小菜鸟，什么开心不开心的，什么情况啊？”

“我们公司最近有一个项目出了问题，公司让我去负责检查程序，修改代码。你要让我去做开发，再苦再累我也愿意，改别人写错的代码，不仅效率不高，我做起来也没有激情啊！”

“呵呵，你这种想法不怎么样，看来我得向你传授一下菜鸟的生存大法了。”

有蔡佳娃这种想法的新手应该会很多，认为公司分配给自己的任务离自己想象的一杯咖啡作伴、一阵思考、一份成功实现的喜悦等场景相差太远。其实，这也不能怨上级，一个刚到公司的人，怎么能事事都如愿呢？

相比来说，蔡佳娃这样的工作算是好的了，有些新人到了公司，上级在一开始甚至都不会分配编程的任务，有可能去做测试或者做售前售后等。

在这里就要发扬菜鸟的精神了，不论上级分配的任务你是否喜欢，是否觉得有意义，都不能有怨言，都要圆满高效地完成。对于菜鸟来说，完成工作任务是底线，也是菜鸟走向成熟的必经之路。套用文艺界的一句话就是：“没有小工作，只有真菜鸟”。

“知道了吧，只有把琐碎甚至‘低级’的工作全部完成好，主管才知道你这个人还是有一定能力的，才能为更具有挑战性的任务做好准备。”

“嗯，可能是我这个人有点太急性子了，要慢慢来是吧？”

“当然不可以急于求成。你想想看人家达·芬奇在创作出《蒙娜丽莎》之前，画了多少个鸡蛋啊！那些鸡蛋绝对是出效果的。”

“呵呵，说得也是啊，我这鸡蛋画得还不够呢。”

以上这种情况是分配的工作不喜欢但是也必须做，而且这些工作是可以胜任的。而下面的这种情况就不一样了，菜鸟在进入职场的时候，可能还会接到稍稍超出自己能力范围的工作任务，这时候适当发扬“不懂装懂”的精神就显得很有必要了。

“师兄啊，果然如你所料，前两天完成那个恶心项目的修改后，主管还比较满意，现在公司终于给我机会做新的项目了。”

“呵呵，那就好好干吧。”

“不过我头更大了，主管分配给我的是一个数据库查询的任务，这可把我郁闷坏了。”

“哦，那是为什么啊？什么问题这么难啊？”

“就是这两张表的查询，一张表是员工信息表（见表 5-1），另一张表是部门信息表（见表 5-2），要求是查询员工平均工资大于 3000 元的部门列表。”

表 5-1 员工信息表

ID	Dep_ID	Name	Gender	Age	.....	Salary
1	3	Tom	Male	26	.....	3500
2	2	Maggie	Femal	27	.....	3500
3	2	Edward	Male	32	.....	4550
4	4	Bill	Male	25	.....	2800
5	1	Penny	Female	23	.....	3800
6	1	Jerry	Male	25	.....	3260
.....	.....	.....	.....	.....	.....	.....

表 5-2 部门信息表

Dep_ID	Dep_Name	Dep_Head	.....	Dep_Size
1	Admin	Jerry	.....	12



续表

Dep_ID	Dep_Name	Dep_Head	.....	Dep_Size
2	Sales	Edward	.....	8
3	Financial	Tom	.....	4
.....	.....	.....	.....	.....

“哦，这个题目要是想做好的话还是有一定难度的。”

“师兄你也知道我数据库方面比较笨一些，不过我当时还是二话没说应下来了，主管说给我半天时间，我搞了一下午，正确的查询结果算是出来了，但是效率太慢了。下午的时候主管正好去开会，也就忘了问我进度，我马不停蹄地弄到晚上还是没搞定。”

“呵呵，看来我们的蔡佳娃还是很要强的嘛。”

“师兄你就别笑话我了。明天主管要问起来我就惨了，你帮帮我吧。我用的是联合检索。怎样才能提高效率啊，要检索的数据有几万条呢。”

“好吧，师兄我就帮你一回，你这种写法访问数据库会太频繁，时间都耗在数据库与 Java 之间的数据传送上了。其实这种情况下要用多层嵌套检索，只用一句 SQL 语句就行了，应该这样写……”

“哦，原来是这样啊。师兄太感谢你了！”

相对于第一种画鸡蛋的情况，这种分配给困难任务的情况其出现几率要大很多。在这个故事中，蔡佳娃这种宁可私下受点苦，也不坦白自己无法胜任的做法还是比较值得肯定的。不要以为是菜鸟就可以有理由不会，菜鸟应该有一种不会也要硬上、打碎牙齿往肚里咽的精神。

虽说要做一个诚实的员工，可是在接受工作任务的时候，就算自己某一方面不行、做不来，宁可自己私下里多花些时间研究，多付出些精力，也不要轻易向上级坦白。很多时候跟上级打交道就像是高手在比武，要避免露出破绽。而且说到底老板也不是道德委员会的，从你的诚实中上级领悟到更多的是：下次直接找别人。

想想看，接下自己不熟悉的工作任务，本身不就是个学习的好机会吗？不然哪里来这么大的动力和压力去研究新问题？宁可人后受罪一些，也要试着人前显贵。不过一定要把握好这个度，实在不能扛的就不要死扛，否则搞不定人前也会很受罪的。

5.1.2 菜鸟不应该自卑

自卑，或许是每个涉世未深的菜鸟在成长中最容易出现的感受。除去那些天才，一般人在进入职场的时候都会需要一个缓冲期，有些人能顺利熬过去并继续向着高手的目标前进；有些人可能会产生很多的负面情绪，从而无限延长自己的菜鸟阶段。

“蔡佳娃，怎么样，上次那个数据库查询的案例完成了吧？”

“多亏了师兄啊，那晚我听了你的方法后又苦战一番，总算是把效率提升上去了，第二天给了主管他还是比较满意的。”

“呵呵，这样很好嘛。继续加油吧。”

“可是，那也不是我的功劳啊，要是没有师兄，这事估计要困扰我很久呢。我是不是比菜鸟还要菜一些啊？”

“看看，怎么又这么自卑了？之前是谁那么斗志昂扬啊，怎么这么快就泄气了呢？”

“当时还没进到这个行业，只是说说空话，现在在职场混了几日，见的东西越来越多，也就越来越发现自己的无知了。”

“这不应该啊，古人都是‘闻过则喜’，你怎么倒越来越消极了呢？看来还得让我给你做做思想工作，去去你的菜鸟综合症。”

在 IT 行业，高手和菜鸟之间的差别可以非常大。很多高手都是菜鸟们仰视都无法看到的高度，而且层出不穷的新技术也在飞速地更新，所以刚刚入行，难免会自愧不如。

不过如果过分感到自卑，心态就有些不健康了，要知道所有的人都曾经是菜鸟，都曾经仰视过别人，慨叹过自己的渺小。不过如果那些仰视和慨叹最后都化成了奋斗的动力，那么你也会有被仰视的那一天。

“蔡佳娃，你应该这样想，比你高深的人也是从你这个阶段过来的，你可能是太着急了，急于让自己成长起来，所以短时间内没有看到结果，就有些灰心了。”

“可能是吧，主要是我觉得自己和他们的差距太大了，境界也差得很远哪。”

“说到底，你比他们少的不就是技术和经验吗？这些都是可以通过自己的努力来获得的。你怎么不想想你比他们有优势的地方呢？”

其实优势和劣势在有些时候并不是绝对的，运用好也可以将劣势转化为优势。作为职场新手的菜鸟也是有一些优势的：

- 新手的思维往往没有像成手那样被一些来自行业或自己的条条框框所束缚，有时反而容易在某些方面有所创新和突破。
- 新手更有冲劲和学习的激情，更容易接受和学习新技术。
- 新手更年轻，有更多的时间和旺盛的精力。

“嗯，师兄，听你这么一说，我心里倒也平衡了一些，呵呵。”

“蔡佳娃，自卑不可怕，关键是如何看待自卑，有些人把自卑变成了对自己的不自信，惧怕失利，不敢再出手；有些人则把自卑变成了不甘命运的战斗。我希望你通过自卑找到自己的出发点，或许这样更能激发你奋进。”

针对职场菜鸟比较容易产生的自卑想法，本书提供以下两种解决方法供读者参考。

### 1. 由浅入深，找回自信

自卑就是觉得自己不行，那么可以试着从最简单的事情开始做，上级分配了任务，稍稍规划一下，先完成简单的部分，比如用户界面的设计等，通过简单的事情增加自信心，然后由表及里，不断加大难度。逐渐培养自己的成就感，认定自己是完全有能力做好的。

### 2. 长痛不如短痛法

一个人再自卑，也不是浑身上下一无是处。这种方法先分析自己哪里不够自信，然后刻意地去加强这方面的锻炼，比如数据库不太熟练，以后每次主管分配任务，抢着干数据库模块，如果对自己口才不自信，那么不论大会小会，逼着自己去发言，积极参与讨论。慢慢地自己的软肋也变成了铁拳，自然不会再自卑。

“嗯，师兄，你说得很对，想想看，我怎么会误打误撞自卑起来了呢。”

“呵呵，你最好能想开。自卑这种情绪在IT行业可是要不得的。你看看那些名垂青史的IT巨星，哪个不是踌躇满志、不羁世俗的。真正的高人根本不知自卑为何物，他们只是不断地努力、失败、分析、再努力。IT界不相信眼泪，没人会同情你的自卑。”

“对对，师兄，幸亏有师兄你为我指点迷津啊，否则我就栽在起点了！”

《论语·里仁》有云：“见贤思齐焉，见不贤而内自省也”，进入到IT职场，所见高人肯定很多，碰到比自己强的，则“见贤思齐”，勇追猛赶；碰到不如自己的，则“内自省”，引以为戒。如此一来，怎么还会有时间去妄自菲薄呢。

### 5.1.3 一叶障目，不见泰山

对于菜鸟而言，除了自卑以外，另外一个需要避免的极端情况就是盲目自大了。相对于面对高手和高深技术的自惭形秽，自以为是的井底之蛙就更加危险了。每个菜鸟都应该尽量避免让自己陷入这两个极端之中，否则菜鸟的头衔将永远无法从头上摘去了。

“蔡佳娃，最近的工作怎么样啊？上次你说自己有些自卑，别告诉我你现在还沉睡在不自信中呢。”

“没有啦，听了师兄一席话，我在后来的一段时间慢慢找回了自信，现在做事很有干劲呢。我觉得我的前途一片光明！”

“这就好，不过我要提醒你一下啊，不要一叶障目，不见泰山。小心刚离了自卑的狼穴，又入了自大的虎口。”

“自大啊？我应该还不是这样吧。”

“醉鬼从来不说自己醉，自以为是的菜鸟可是跟不懂装懂的‘高人’一样遭人厌的。”

在IT职场中，“一叶障目，不见泰山”指的是这样一种情况：通过认真学习，掌握了一些知识，也做出了一些成果，但是从中获得的成就感过高。同时由于对于IT这个行业认识非常局限，基本上是“管中窥豹，只见一斑”，便错误地产生了“普天之下，莫非王土”的气概。

“一叶障目，不见泰山”的问题可以用以下几种现象来说明。

#### ● 现象1

学习一门语言的初期，掌握了编程语言的基本用法，写出了一些应用小例子，就觉得再复杂的东西都可以用循环分支加判断写出来。这种“剑指青天”的豪气无可非议，但这种认识就显得很无知了。

比如学习Java，只学习简单的语法和线程异常之类的知识是远远不够的，还要深刻地理解面向对象的思想 and 了解面向对象的设计模式等相关的知识才能真正胜任开发人员的岗位。

#### ● 现象2

和现象1不同，在现象2中那些被“障目”的人对于一门技术掌握得还是可以的，但就是因为对于某一种技术过于自信，认为是无所不能的，到哪里都要用，而看不到其他技术在某一领域的绝对优势。

比如用Java开发一个学校教学信息管理系统，对于数据库的操作只有一句“select \* from...”，

其他对数据的处理全部放在 Java 这边用循环来完成。系统做完以后，输入进去 3 名老师、5 个学生及 8 门课的信息，测试效果非常好，速度也很快。而随便找一个学校的真实数据（比如老师 100 名，学生 2000 名，课程 120 门），系统立刻运行瘫痪。原因就是太相信 Java 无所不能，不愿意深入学习 SQL，从而看不到 SQL 在数据库操作方面的优越性。


### ● 现象 3

很多人认为自己知识掌握得很牢固，技术能力也很强，就算是高手了，但是却忽略了很重要的一点——实战经验的积累，而往往高手和菜鸟之间最大的差距就在这里。

比如开发一个手机端与服务器交换数据的项目，如果曾经编写过手机向服务器发送“Hello World”字符串之类的程序，做这个项目应该感觉不会太难。但是在实际开发过程中却发现一张图片传到服务器后只剩下一条线。原因就是手机在向服务器发送数据的时候有一个上限，所以有经验的开发者都会先开发出一个分组发送数据的功能类库，然后再在项目中调用以发送超过上限的数据。

综合以上 3 种现象，现象 1 中忽略的是一门编程语言的思想和设计模式等深层次的内容；现象 2 中被“一叶”遮住的是其他语言技术的博大精深；而现象 3 中看不见的“泰山”则是宝贵的实战经验。

以上 3 种现象是有共同点的，那就是：初步掌握了一门技术的基本知识后，从纯理论上进行推导认为用这些知识可以开发出所有的应用（如果真是这样，大家都直接用图灵机算了）。而且对于这个理论推导的结论过于执着，不再深入学习，直接抱着这种信念用很基本的技术去开发所有的功能，开发完后再用很幼稚的模拟数据进行测试，运行无误后就认为是开发成功了。结果拿到生产环境中却无法让客户满意或根本就不能正常运行。

 **提示** 读者朋友们请注意，如果平时已经有符合上述论断的言行就要十分谨慎了，否则不但影响职业生涯的健康发展，还会贻笑大方。

“蔡佳娃，我来问问你，有没有做完一个模块兴奋过头的时候？有没有对于自己目前掌握的技术过于自信的时候？有没有不想听同事或前辈说话或者一直想打断的时候？”

“这个我倒是没有，就是有时候稍微得意一些，不过我们公司有一个新来的和你说的倒挺吻合的。”

“这就是很典型的‘一叶障目，不见泰山’了。很早我们的老师就会讲，一个人所知道的知识就像一个圆，知道得越多，未知的事物就会越多（见图 5-1）。你想想看，你们公司那个人如果真的对自己圈外的知识都掌握了，那么他的知识圈该是多么小得可怜啊！”

“说得太对了，我们公司那个人根本就没那么厉害。或许他并不知道什么是真正的厉害吧。我跟他说话的时候他总是没等说完就打断说‘嗯，我懂’，好像很怕别人以为他不懂的样子。”

“是啊，身为菜鸟这么做可是很不招人喜欢的。天外有天，人上有人，他是应该知道这个道理的。”

学之深，路之艰。就像故事中的知识圆圈模型一样，学到的知识越多，越会发现自己的无知。况且工作中除了技术上的问题要解决，客户有时略显刁难的要求也需要费一番脑筋。所以不要以为做过一两个“精彩绝伦”的项目，就可以自称为大牛了。如果大牛都可以速成的话，那么 IT 还不成了豪杰四起、群雄逐鹿的乱世了。

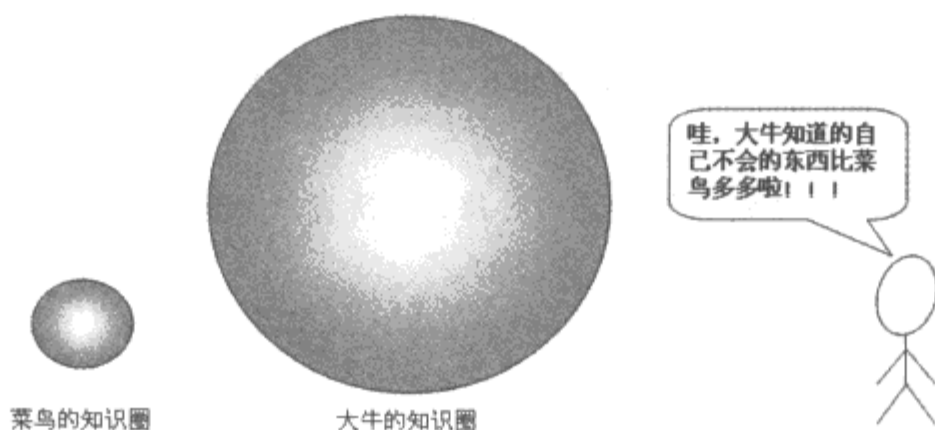


图 5-1 知识圈模型图

“蔡佳娃，你知道一个项目经理或主管是如何为自己的团队挑选成员的吗？”

“不知道，那师兄你说是怎么做的？”

“我曾经和我们公司的一个项目经理聊天，谈到招聘人才时，我问他为自己的团队挑选成员的原则是什么。”

“那位项目经理是怎么说的啊？”

“他说只有一个条件：听话。他说很不喜欢那些一瓶子不满、半瓶子晃荡的冒尖员工，尤其是那样的新员工，倒不是个人的好恶，而是这些人不管是不是真的很厉害，到了团队只会影响大家，还不如一个忠诚听话的员工呢。”

“是啊，看来我今后要尽量避免成为那样的人。”

“是啊，要搞清楚自己新入职，再牛也是个后辈，经验和教训都少，所以努力让自己进步是首要任务，盲目自大是万万不可的。”

明太祖朱元璋起事的时候，谋士刘伯温就曾向他献策：“高筑墙、广积粮、缓称王”。如今这个建议除了第一个“高筑墙”之外，其他六个字也非常适合刚刚进入职场的菜鸟用来自勉。不要总想着“毕其功于一役”，在IT行业闯荡，要有“一将功成万骨枯”的气概才行。

## 5.2 锐意进取，菜鸟无敌

5.1节谈了谈菜鸟如何在新入职的时候摆正自己的姿态，摆好姿态后，就要开始冲锋陷阵了。本节将会为广大锐意进取的菜鸟们出谋划策，希望可以帮助广大菜鸟朋友一飞冲天，成长为一个IT江湖的金凤凰。

### 5.2.1 既是初生牛犊，就别怕虎

俗话说“初生牛犊不怕虎”，IT菜鸟进入职场，本身就是站在最低点，再没有什么可以输的。因此大可以做个初生牛犊，靠着一份不怕虎的精神仗剑闯天涯。

“师兄，你也是从菜鸟过来的，现在我正处在这个阶段，你也给我支几招，让我做个勇往直前的菜鸟吧。”

“难得你斗志这么高，我结合我当年奋斗的经历，给你提几个建议吧。”

“嗯，成功人士的经验一定是很值得参考的！”



“首先，对于工作必须要有热情，不然你没有办法让自己心甘情愿地为之废寝忘食。软件开发更多需要的是脑子的灵光闪现，在充满激情的状态下，大脑更容易刮起头脑风暴，卷出一堆堆好点子和创造性的想法。”

“说得也是，不过当工作变成一种重复的时候，热情自然会慢慢消退，就像之前我负责给公司一个失败项目找错修改的时候，真是很没意思的。”

“对工作的热情没法强加于人，你需要试着让自己从重复或略显枯燥的工作中寻找乐趣和动力，如果没了激情，那么你就跟失业一样感到空虚。”

一个项目的开发在给人带来成功的愉悦之前，多半会先经历那些失败的沮丧。所以若没有足够的工作激情的话，很难让自己免于被失败击倒，从而大大降低了工作效率。

说到经历失败，坚韧意志的培养对于一个菜鸟来说也是非常重要的。有过编程经验的人应该都清楚，就算是一个小小的程序，开发起来往往由于不容易注意到的细节，也会造成令人抓狂的bug。而解决这些棘手问题的唯一办法，大概就是耐心和坚持了。

菜鸟一开始接触的项目任务虽然都不会很大，也要注意在工作中保持旺盛的斗志和不轻言放弃的坚韧。这样在接到越来越大的项目时，在心态上已经成熟到可以应付得了任何可能出现的非正常情况，自然会越战越勇。

“蔡佳娃，关于工作激情我们就谈到这，毕竟对待工作就像对待恋人一样，热情也是需要培养的，关于恋爱的法则你可以拿来用在工作上试试。”

“啊？师兄你可真会比喻。我们下面谈什么啊？”

“下面说说另一种菜鸟锐意进取的方式，那就是做别人不愿意做的事。”

“这具体指的是什么啊？”

“很多方面啊。比如说某个项目中有一个模块大家都不熟悉，没人愿意做，你来做。团队里要找个人去给客户做需求分析和项目报告，没人愿意去，你去。客户使用软件的时候出了问题，没人愿意去调试，你去。”

“师兄，这么冒进，万一做不来怎么办哪？”

“你怎么总是还没开始就先想到失败了怎么样呢？你对自己的失败显得很自信呀。你要想想这些事要是做成了，今后在公司你就是这方面的权威了。就算失败了，至少你尝试过，你也能够弥补自己在这方面的空白。”

勇于去做别人不愿意做的事，主要就是靠这种拼劲和胆量。很多时候不是大家都不愿意去，而是在等着钦点或者推举，在这个“万马齐喑”的时刻，毛遂自荐般地挺身而出就非常令人印象深刻了。所以很多时候菜鸟的机会都是“抢来”的，而不是天上掉下来的。

“抢”下别人不愿意做的事，并不一定是自己擅长的方面，这种破釜沉舟的行为很容易将自己放在不得不艰苦奋斗的局面之下，断掉自己的后路是很需要勇气的。很多时候不是菜鸟成长的环境不好，而是菜鸟本身给自己的压力太小。

同时，揽下别人不愿意做的事，做成了固然好，做坏了至少也是一次教训和经历。爱迪生发明灯泡的时候失败了一千多次，别人讽刺般问他的时候他说：我并没有觉得失败，只是找到了一千多种行不通的方法而已。不管成功与否，都为之付出了很大的努力，而且研究的可能还是自己

从未接触的新技术，收获或多或少还是有的。

“蔡佳娃，下面我就要提一提菜鸟在锐意进取的时候需要注意的一个问题。”

“哦，那是什么啊？”

“在进行自己的工作任务时，不要所有的问题都叫主管来事事躬亲，在一定程度上有自己的主见才行。”

“那得事先把要做的需求分析都了解好吧。”

“主管分配任务后，弄清楚自己该干什么。然后埋头自己做，没有大问题，不需要再去麻烦主管问这问那。软件开发更多的是一种创造性的活动，就像盖房子，如果主管精确到每一块砖的去处都详细告诉你的话，那么你也只是做了做搬砖的力气活，整个房子在思想上是不属于你的。”

“是啊，总问问题的话，主管也会觉得你这个人没什么主见，什么事都拿不准。”

总是追着上级主管不放，询问项目的某个模块是选A方案还是B方案，这种人不一定是技术不行，只是做事太求稳妥，总想保证万无一失。不过这种人在主管看来，要么做事太不自信，要么根本没有经验，要么没有自己的主见。

身为菜鸟，入行谨慎些是没有错的，但是要对自己的能力有信心，要试着以自己的思维去理解项目需求，用自己的思维去模拟实现项目需求功能。这样每做一个项目，对于该项目的架构和技术的落实情况都会有充分的把握。经验就是这么积累起来的。

“蔡佳娃，听我这一番讲述，你是不是觉得我希望你们菜鸟个个都具有很强的侵略性啊？”

“真的有点啊，菜鸟不是应该更谦卑好学一些吗？”

“我的想法和你的也不矛盾啊！你所指的是工作态度问题，而我的意思也很简单，就是在工作上要锐意进取，刚入行的菜鸟和得道高人相比，唯一的优势就是年轻有冲劲。这如果再不好好加以利用的话，成为高手的机会就变得很飘渺了。”

“话是这么说，但是跑得太快，摔跤的几率也会更大一些的。”

“在担心犯错吗？你知道做菜鸟的好处是什么吗？就是‘三不怕’。”

“哪三不怕啊？”

“不怕挑战，不怕犯错，不怕挨骂。所以要想早日成为高手，就要牢记这三个不怕，做一个展翅高飞的菜鸟。”

俗语说得好：光脚的不怕穿鞋的。不像那些打拼数年的前辈们，刚刚步入职场的菜鸟，就相当于一个赤脚的人，没有资本，没有顾忌，没有什么可以失去的东西。需要做的就是坚定自己的立场，将“三不怕”的精神发扬光大，让自己迅速成长为一名高级菜鸟。

当然，就算是初生牛犊，也不能把老虎惹急了。IT菜鸟在努力提高自己的时候，也要防止快犊破车，斗志高昂可以，但是盛气凌人就很不理智了。

“蔡佳娃，我给你讲个生炉子的故事，你就大概明白菜鸟成长的过程了。”

“哦，生炉子还和这个有关系吗？说来听听。”

“生炉子的时候需要用的材料是纸板、小树枝和炉子最终的主要燃料大木块。生火的时候如果三者之间的缝隙大，那么纸板和树枝烧完了，木头还只是被熏黑。”

“是啊，纸板、小树枝和木头之间没有配合好，木头没有得到足够的热量。”

“正解，所以要合理搭配纸板、小树枝和木头的数量和疏密程度，使纸板能较快地引燃小树枝，而小树枝也有足够的热量让木头燃烧。这样炉子就算生起来了。你能明白其中的道理么？”

“哦，师兄，我明白了。菜鸟的成长就是这样。燃烧木头是菜鸟最终要达到的境界，而在此之前的每个阶段都要适当地连接起来，不可火势太猛，否则造成后劲不足，对不对啊师兄？”

“嗯，看来我的小师弟的确是个悟性极高的大菜鸟啊！哈哈。”

总结一下，菜鸟在锐意进取、向高手迈进的时候，需要遵循和注意的地方有以下几点。

- 工作要有激情，毅力要坚定。
- 敢做别人不愿意做的事。
- 不要事事咨询，要有一定的主见。
- 光脚的不怕穿鞋的，要有自己的立场。
- 当心快犊破车，后劲不足。

### 5.2.2 勤于学习，落后就要挨打

前面谈论了 IT 菜鸟在工作中应该如何积极进取，提升自己的话题，本节将谈谈工作之外的一种修炼方式——学习。

每个身在职场的人都应该有危机感，就连刚起步的菜鸟也一样。IT 世界的技术日新月异，淘汰和更新飞快，今日的潮流技术有可能成为明日黄花。而且就算是主流的技术，如果掌握的深度不够，仍然是不能成就于江湖的。

所以要想跟上 IT 产业的前进速度，或者深入地掌握一门技术，不间断的学习就很有必要了。而在如今的互联网信息爆炸时代，学习的方式也是多种多样的。

“蔡佳娃，作为一个 IT 人，危机意识是必须要有的啊！”

“啊？危机意识，这个我倒没有想过，像我这样的新手也要有吗？”

“怎么没有啊。看看当今的 IT 行情，难道你没有想过自己现今的一技之长有可能在未来会被更新的技术所替代？你看看周围，公司的前辈或是和你同样菜的同事有没有人在暗自努力，想在你们共同的技术领域做个只手遮天的人呢？”

“这倒是啊。那应对危机有什么办法呢？”

“古语有云：‘生于忧患，死于安乐’。有危机意识，就要通过自己不间断的学习把危机变成转机，变成前进的动力。”

《三国演义》中董卓作乱，十八路诸侯起兵共同讨伐董卓，孙权之父孙坚为前锋，率精兵攻汜水关，斩华雄副将于马下，作战勇猛。然而在与敌人相持了几日之后，却因各怀鬼胎的后方诸侯迟迟不将粮草运来，致使军心不稳，被华雄夜袭营寨，损兵折将，败阵而归。

论战斗力和谋略，开创了江东基业的孙坚绝对要胜华雄一筹，可是最后偏偏输在粮草上，让人饮恨。其实 IT 菜鸟在职场上的闯荡也是如此，往往冲劲十足，等杀到敌前，或者杀到正酣，发现没有了后劲，结果必然是“出师未捷援先失，长使菜鸟泪满襟”。

“那师兄你说说看，我该怎样在让自己在快马加鞭的时候提升自己的后劲呢？”

“方法有很多，不过最实用的就是去啃书了。”

“像我这样的菜鸟应该啃些什么书好呢？”

“首先当然是你自己从事领域的技术书籍了，然后读一些大公司开发者社区的文章，如 IBM、Sun、Oracle 等。同时，也可以多读一些权威组织的相关学术论文，如 ACM 和 IEEE，多看新闻了解新技术的发展方向。总之这种学习是很广泛的，对于你加深自己对行业的了解和深入研究技术都很有帮助。”

“嗯，你说的这些东西我还真是很少接触呢，看来要尽快补上这一课。”

“学习的内容不要局限在技术上，很多其他方面的书也是需要看的。比如一些讲述 IT 风云人物的传记，或者是与你行业有关的知识，比如你们公司经常做手机业务，那么对于市面上各种主流手机的功能和特点就要有所了解。”

“看来要学习的东西还真是不少啊。”

“那是当然。你要记住，在 IT 界，落后就要挨打。所以学习不仅是在提升自己，更重要的是在保证自己在行业中的生存机会。”

荀子的《劝学》有云：“吾尝终日而思矣，不如须臾之所学也”。说的意思是整天的思考不如片刻的学习，这话用在严谨实干的 IT 行业可谓非常适合。虽说是学海无涯，但是如果学都不学的话，那么就真的是苦海无边了。

IT 行业竞争激烈，对于菜鸟来说更应该未雨绸缪，绝不可临渴掘井。一直保持在学校读书学习的习惯不要丢弃，才能让自己在进步的道路上粮草充分，后劲十足。

**提示** 说到要去读 ACM 和 IEEE 的学术论文，很多读者会觉得奇怪，我辈企业派人士也要搞学术研究吗？其实多读学术论文对职业成长是很有裨益的，这在后面的章节中会详细讨论。

“说到啃书学习，有一点我要提醒你。这是一个很多人都容易犯的错误。”

“师兄，那是什么啊？”

“在看技术书籍的时候，一般都会有大量的程序代码和随书光盘，你不要看一遍书，理解一下代码，把随书光盘里的代码部署到机器上看看效果就以为完事了。那么做给你留下的印象还不如看一场好电影来得深刻呢。”

“那要重新做一遍吗？”

“最好是看完后重新做一遍，代码是人家写出来的，书写风格和编程习惯都不一样。你要用自己的思维方式和编程模式去完成同一个项目，这样才算是把书中的知识都掌握了。”

“先看一遍，等用到的时候直接把代码复制然后进行改动不也可以吗？”

“完全可以。不过如果你去看病，一个医生给你把把脉，然后翻翻书，再观察一番，再翻翻书，之后给你确定病情，最后再翻翻书把药抓了；另外一个医生把完脉，询问几句，直接对症下药。两个医生都能保证把病治好。那么下次看病你会选哪个医生？”

“肯定不会选那个离不开书的医生了。”

“一样的道理，你应该明白自己要做哪种人了吧？”

在这里提醒那些不喜欢自己实现书中代码的同学，IT 行业可是不存在代码观察师这个职位的，一般也不会为部署项目专门设置一个职位。所以还是不要想着将自己的知识分散在各个书中以便需要的时候再去翻阅，霸道地集中在自己脑中才是正确的方法。

啃书的时候，尤其是在啃含有大量实例的项目书籍时，一定注意不要仅仅满足于将项目看懂、把代码弄清楚。菜鸟缺的不是观察经验，而是动手经验，看懂了不是自己的，只有做出来的才是自己的。否则就会像故事里的那位医生一样，离不开书了。而且自己没有亲自完成过的内容是很难灵活运用，从而举一反三、触类旁通的，那样学习的效果就大打折扣了。

### 5.2.3 菜鸟应该懂得的几件事

在奔向高手的道路上，菜鸟除了要发扬初生牛犊一往无前的精神和养成勤于啃书的习惯之外，还有一些做事的方法和技能要掌握以便让自己的高手之路更加平坦。这些主要包括以下几方面：

- 编写文档材料的能力
- 做 Presentation 的能力
- 规划的能力
- 遇到困难的解决方式

#### 1. 编写文档材料的能力

文档材料的编写，对于还在学校的大学生和职场新人来说，大概也只局限于了解如何编写项目的开发文档，如项目的需求分析说明书、概要设计说明书、详细设计说明书等。但是在 IT 的职场中，需要编写的文档材料种类非常多，而且随着职位和技术的提高，所需要编写的文档材料还会更多。

“蔡佳娃，你知道 Coder 和 Developer 的区别是什么吗？”

“是什么啊？资历和技术水平的差异吗？”

“有这些方面的差异，不过还有一个比较重要的差异就是编写文档材料的能力。”

“是不是跟在学校写实验报告那样的？”

“怎么会一样呢？看来大学让你对文档这个概念产生了很难消除的误解。IT 行业的文档是用来创造财富的，你认为在学校写出来的文档能不能拿给上级或客户换饭吃？”

“呃，那还真不一样。不过也要为客户编写文档材料吗？”

“客户是上帝，难道给上帝写个材料很过分吗？现在你还不觉得，等你当上了项目经理或主管，和客户沟通的时间就更多了，而沟通的主要方式之一就是写材料了。”

总结一下，开发人员需要在工作中编写的文档大概有以下几个方面。

- 技术文档

包括程序开发文档、项目的概要设计说明书、详细设计说明书等。

- 客户沟通方面的文档

包括给客户的产品使用说明书、需求分析说明、客户反馈文档等。

- 管理性质的文档

管理性质的文档一般是职位在项目经理、Team Leader 之后需要书写的，包括项目预算报告、项目计划时间表、项目验收鉴定材料等。

- 其他方面

这些文档就比较自由了，包括一些项目程序的注释文档，也可以包括应邀写的技术介绍、经



验分享文章，或自己写的博客甚至书稿等。

## 2. 做 Presentation 的能力

与书写文档材料要求的是文字功底相比，Presentation 主要指的就是口头的表达能力了。在以技术和思想称霸的 IT 行业中，Presentation 和书写文档的能力并不是决定性的，但是这些技能却像是润滑剂，掌握了它们，就会显得比他人更有优势。

“师兄，今天我们公司开会，说要找个人明天去客户那里做一个产品说明，顺便搞一下用户培训，我看没人愿意去，于是就自告奋勇了。”

“呵呵，不错。就是要有些闯劲嘛。”

“可是这个产品说明是干什么的啊？我觉得对于自己提高没多少用啊？”

“错，在你的职业生涯中，会有很多机会需要你将自己的想法和做法用语言或图表表达出来以便交流看法和说明问题，这是促进团队合作和加强和客户交流的重要渠道。像你明天要去做的产品说明和用户培训就是这样，它们都属于 Presentation 的范畴。”

“虽然我不太擅长这个，不过 Presentation 有这么重要吗？”

“那是相当重要啊！如果 IT 行业只重技术不重交流的话，早就退化了。你可能有能力，但是如果你的语言表达能力不行的话，那也只能是茶壶装饺子，有话道不出了。”

一般情况下，Presentation 主要包括以下几个方面的内容。

- 客户沟通

主要包括对客户进行的产品展示介绍、用户培训、项目进展情况说明等。

- 经验分享

主要包括和开发团队成员进行技术上的沟通，分享既有成果、共同进步，或是团队内部共同探讨如何解决一个技术问题。

- 新技术介绍

主要是应邀对公司内部或外部的开发人员介绍、培训自己所学习或研究的新技术，其目的主要包括：帮助自己公司的内部技术人员快速掌握新技术或对外推广自己公司或组织的新技术。

想让别人看到自己的优秀，就要懂得如何去展示，本节中的编写文档和做 Presentation 都是展示自我的重要途径。而且像编写文档和做 Presentation 的能力，都是随着经验的丰富而逐渐提高的，所以每个人在进入职场的时候，都要注重自己技术之外的其他相关能力的锻炼与提升。

## 3. 规划的能力

在 4.3.2 节中，笔者已经与各位读者讨论了定目标的意义和学问。随着对职场的深入了解，本部分将继续探讨规划的重要作用。这里所要讨论的规划，将不仅限于规划目标。

“蔡佳娃，还记得你在学校的时候我们谈过的关于目标的话题吗？”

“记得记得，定目标不在宏伟，贵在不变。”

“嗯，今天我们将深入地谈一谈规划的问题，首先温习一下目标。不知道当初我们谈了目标之后你有没有回去为自己定下一个目标呢？说说看。”

“有啊，不过，师兄你应该听过初恋不懂爱情吧？我上次定目标就跟初恋差不多，定的目标很

青涩，也很无知，我还是不要说了。”

“呵呵，其实这很正常，定目标完全是自己的事，我可以不问。不过你这样也算正常，毕竟当初在学校对这个行业的了解只是听来的和看来的。等真正进入到这里面，才会对自己在这个大圈里面所扮演的角色有个切身的体会。”

“是啊，我当初的目标就是不太实际，所以才不好意思讲的。”

“所以我现在跟你提一下，既然你在学校定下的目标已经很不切实际了，那么就要赶紧审时度势，重新为自己定下宏图远志了。”

“蔡佳娃，其实不仅我们之前讨论的目标是一种规划，对一天的工作安排、对一个项目模块的工作量分配等都算是规划。”

“是啊，有时候一天的工作如果没有个日程表，一天下来竟然也会忙，但却不知道在忙些什么，根本没什么收获。”

《礼记·中庸》有云：“凡事预则立，不预则废”，做任何事都要有计划和准备，这样才可以做成，否则就很容易失败。每个进入 IT 职场的新人，都应该为自己做好合理的职业生涯规划，否则没有方向的奋斗将是很难有所建树的。

《大学》里面还有这样一段话：“知止而后有定；定而后能静；静而后能安；安而后能虑；虑而后能得”。这段话的意思是：知道自己所能达到的目标才能够坚定意志（止代表所能达到的目标或境界），坚定意志后才可以内心平静下来而不是浮躁不安，平静下来才能够深思缜密，考虑周全，深思熟虑之后方可有所收获。

这段话的后面几句就是大家都比较熟悉的“修身、齐家、治国、平天下”了。祖先们对于成功的阐述确实透彻精妙，在此也把这句话送给各位读者以共勉。

#### 4. 遇到困难的方式

从事任何工作的过程都不可能是一帆风顺的，因此在遇到困难的时候能够很好地面对困难，找到解决困难的方法也是非常重要的。

“其实不仅要规划好自己的工作，也要学会安排自己做事的方式。”

“哦？那应该怎样安排呢？”

“比如在工作中要分轻重缓急，在开发一个项目的过程中，很容易由于一些隐蔽的编程错误而陷进去。所以在遇到这种棘手的困难时不可钻牛角尖，不要因为一味地坚持‘黄沙百战穿金甲，不破楼兰终不还’的精神，而耽误了每天的工作计划。”

“是啊，我深有感触，当被那个找不出来的错误折磨得要死的时候，就已经在浪费时间了。”

“的确。有时这种情况下跳出来换个思路，或者放下来去做别的事情。没准就会‘梦里寻他千百度，蓦然回首，那人却在灯火阑珊处’呢。”

上述场景正是笔者早年工作中常犯的错误，其实当遇到一个问题暂时解决不了时不应该一直死磕，可以休息一下或先做其他工作。一般大脑经过一段时间调整后都能有新的 idea，跳出原来思考的僵化模式，反而有助于问题的解决。读者朋友们在实际工作中也应该注意这点，避免因为一个小问题而大大拖慢了项目的进度。

## 5.3 知足常乐，健康心态

锐意进取，一往无前，的确是每个菜鸟在入门时期都应该有的工作态度。但凡事都要有个度，随着时间或是技术的推进，每个人都会有走不下去或暂时走不下去的情况。这个时候再去锐意进取只怕会事倍功半了，所以知足常乐就是此时的工作态度了。

### 5.3.1 总有你达不到的高度

虽然是事在人为，但是无论在什么行业，“人比人，气死人”的说法也是个很实在的道理。其实不光是和别人比，即使是一个人，不管如何努力，总也会有自己达不到的高度。

“师兄，好久不见！”

“嗯，蔡佳娃，最近去哪里了？怎么很少在网上见到你啊？”

“呵呵，上次和你一起吃饭谈的那些东西，我回去好好想了想，觉得是该好好规划一下自己的人生目标了，趁着年轻有干劲多闯闯。然后我就发扬你教我的‘初生牛犊’精神，一直在刻苦努力着呢，所以就很少上网聊天了。”

“不错啊。原来是在用功啊，值得鼓励呀！不过既然说到这我就要提醒你一下，刻苦用功是好事，不过你也要对‘山外有山，人上有人’这句话有充分的理解啊。”

不管多么豪情万丈，胸怀经天纬地之才，也有英雄迟暮的时候。或者像周瑜那样遇到比自己略胜一筹的孔明而败下阵来，或者像诸葛亮那样输给自己无法避免的老去而饮恨陨落。总之，需要在自己的凌云壮志和能力极限之间找到一个平衡点。

菜鸟在IT行业打拼，侵略性和攀比心不要太强，靠着年轻的冲劲和胆量，确实能做出一些成就，但是不要太年少气盛。当到了自己所不能再上升的高度时，就应该暂时停下脚步，学着知足常乐，乐天知命。

“师兄啊，既然每个人在自己的行业中都会有无法达到的高度，那么他们在走到自己的极限时会选择怎样的出路呢？”

“这个问题问得好啊，在面临自己所能达到的最高点时，有些人还是不服气，继续攀登自己的人生高峰，这种人的精神固然可嘉。不过既然我们说的是客观存在的最高点，那么这种努力就有些不可取了。”

“是啊，那第二种对待极限高度的方式呢？”

“第二种就是要有健康的心态，既然我已经达到自己的极限高度，那么我就在这个高度定居，享受一下这个高度的风光，知足常乐。”

“这种方式还是不错的，至少比第一种要幸福一些。”

“还有一种方案就是既然我在这个领域的能力达到了极限，那么我就转头下山，再去找一个方向去攀爬，等达到那个方向的极限高度后，再寻找另一座山峰供自己攀爬。（见图5-2）”

“嗯，看来这种处理方式是最好的。”

“当然了，不过你离着自己的极限高度还很远，还要继续努力攀登，我只是提醒你每个人都会有这样的时候罢了。别等到了那个时候还学第一种方法做无用的努力，那就是竹篮打水了。”

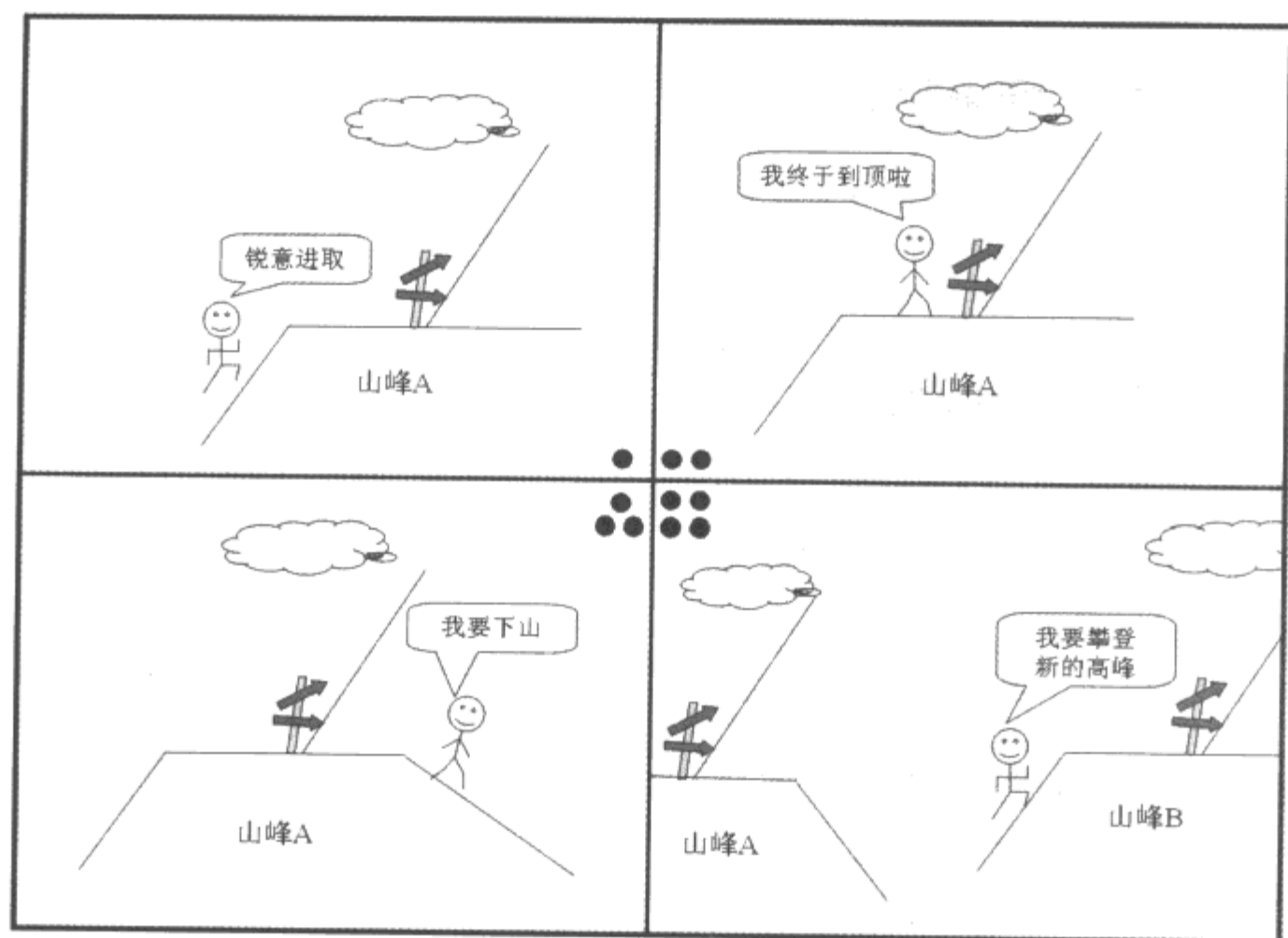


图 5-2 攀登新高峰

或许在谈到如何对待自己人生极限高度的问题上，中国大学生都非常熟悉的李开复可能是个最好的榜样。被誉为“打工皇帝”的他，为自己的人生创造了一个又一个的高度。1998 年李开复加盟微软，出任微软亚洲研究院院长，之后又任微软全球副总裁。随后，李开复离职微软，加盟谷歌任全球副总裁，凭借着自己的才能又将谷歌中国打造成一个年轻的帝国。

然而，加盟谷歌四年之后，李开复拒绝了谷歌再一次的邀请，选择了离开，他认为谷歌在他的带领下已经在中国站稳了脚跟，再也不需要他了。自己也不能再为谷歌做什么了，而他也有其他的事情要做，而这件事就是创办创新工场帮助初出茅庐的大学生创业。

故事中的第三种方式指的就是李开复对待人生高度的态度，完成了一座山峰的攀登，李开复走下山来，继续另一座山峰的攀登。这或许是对待极限高度最好的办法了，读者朋友们或可借鉴。

### 5.3.2 职场爬山论

在职场打拼就像爬山，“无限风光在险峰”，但是能够领略无限风光的人却是少之又少。倒是有限的风光有时也会同样秀美，而且还是通过努力攀爬得到的高度。下面将以爬山为例向读者朋友介绍 IT 职场中知足常乐的意义。

“师兄，你几天前还让我锐意进取，纵横捭阖呢？怎么今天又让我知足常乐，乐天知命了呢？是不是前后矛盾哪？”

“不矛盾，这正体现了中国儒家文化极力推崇的中庸之道，等我给你讲个例子就明白了。既然跟你提到了攀登高峰，我们就把一个人的 IT 职业生涯比作爬山（见图 5-3）。首先从山脚下出发，当然是锐意进取，勇猛攀登，你现在就处于这个阶段。”

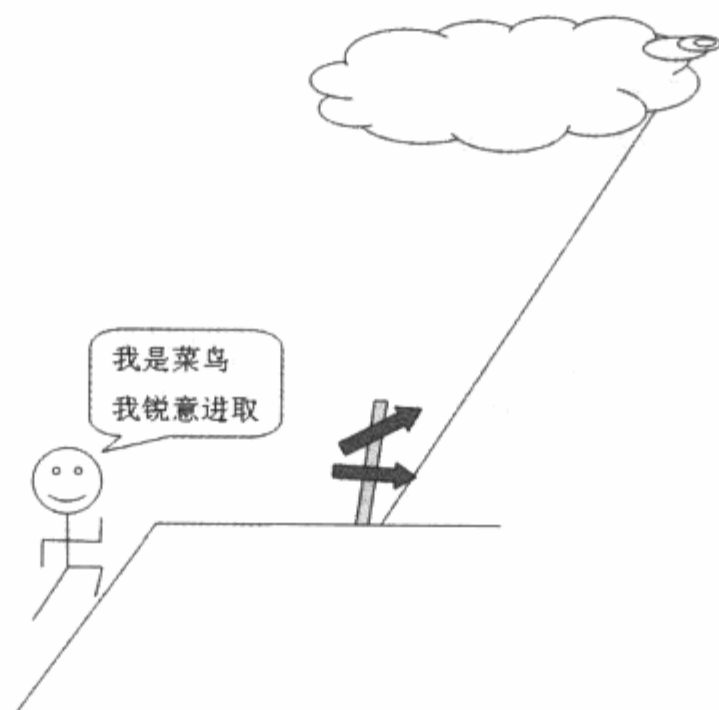


图 5-3 锐意进取阶段

“嗯，我现在就正在征服这座山。”

“但是，随着你越来越往上爬，慢慢地，你的身体和心理就会接近极限。”

“大概是这样吧。越往上爬，面临的条件就越艰苦，所需要付出的就越多。”

“所以你可能会选择停下来，你会发现此地也是风光绮丽，和自己理想中的顶峰风景相比虽然差一些，但至少已经不枉来此一爬了。（见图 5-4）”



图 5-4 知足常乐阶段

“是啊，只要努力地攀爬了，都有权利去享受应得的美丽。那我接着干什么啊？总不能站在那个山头当雕像吧。”

“接下来你所面对的就两条路了：第一条是享受着既得的成就，知足常乐。选择一条平缓的山路，这条路或者是蜿蜒向上的，或者是平绕山间，又或者是娓娓下山的。不管是上下山或是环绕山腰，都是成功的，因为能到这个岔路，都算是付出了努力的。”



“那第二条路呢？”

“第二条路就是在身心到达极限无法继续攀登的时候，在岔路休整一下自己，就像第一条路那样。等到一两年之后，再重背行囊，继续攀登。（见图 5-5）”

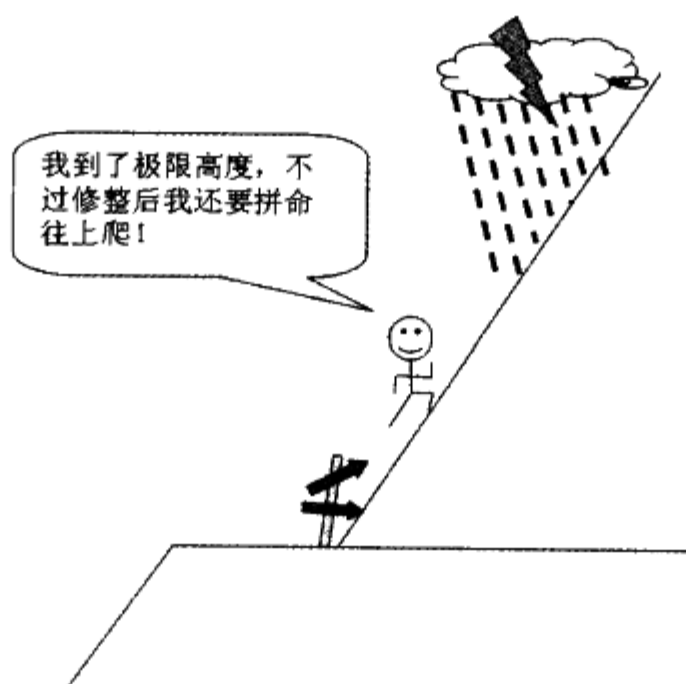


图 5-5 继续攀登阶段

“知足常乐”语出《老子·俭欲第四十六》：“罪莫大于可欲，祸莫大于不知足；咎莫大于欲得。故知足之足，常足。”意思是说再大的罪恶也没有比放纵欲望更可恶的，再大的祸患也没有比不知满足更严重的；再大的过失也没有比贪婪不知满足更离谱的了。所以知道满足的人，永远是觉得快乐的。

其实，IT 职业生涯这座山的高度，既可以是时间长短的标尺，也可以是技术深度的标尺。

- 表示时间长短

年轻的时候从山脚出发，凭着牛一般的冲劲和火一样的热情，向着自己的目标前进。但是随着年龄的增长，自己的身心发挥到了极限，只好停下脚步，选择知足常乐。不过就像图 5-5 所画的那样，知足常乐一段时间之后，还可以继续锐意进取，继续书写着自己拼搏的历史。

- 表示技术深度

刚刚踏入 IT 行业，凭着自己的技术能力，慢慢地向着山顶进发。期间通过不断学习，很多事情是努力跳一跳也能够到的。但是人各有别，总有自己不擅长的东西，做不到就不要强来。选择一条知足常乐的道路，也不失为一种职场生存的良好方式。

### 5.3.2 做最好的自己

当代著名作家刘墉曾经写过：“因为年轻，所以流浪。”的确，在年轻的时候如果不趁着自己有精力和动力去闯荡拼搏，为自己赢得一个人生的良好归宿，等到年老的时候再去流浪就真的可悲了。

“蔡佳娃，我要告诉你的就是，在你刚刚踏进 IT 这个行业的时候，需要靠自己的努力去争取，去为自己赢得一个不错的职位和可观的薪水。你如果不努力，谁都帮不了你的。”

“嗯，对，师兄你很早之前就是这么对我说的。”

“但是，我们都深知，自己这辈子再乐观地估计，也不大可能成为名垂青史的 IT 英雄了。所以你没有必要对自己期望过高，做最好的自己就行。身在紫禁之巅的风景是很高妙，但是也有着高处不胜寒的落寞与悲凉。”

“师兄，你说的话我都记下啦！”

每个人都有享受自己的方法，知足常乐不等于不求进取，不是知难而退。每个人的智商和能力有别，强迫自己去做无法完成的事是毫无道理的，也是不可能取得成功的。IT 世界的竞争再残酷也不会生灵涂炭，万物倾颓，所以只要最大限度做最好的自己就能在其中为自己赢得一席之地了。

## 5.4 菜鸟何以菜，大牛何以牛

本章一直在介绍如何让一个初窥门径的菜鸟快速健康地成长为一个技术大牛，那么菜鸟到底菜在哪里？大牛又牛在哪里？从菜鸟成长为一个技术大牛需要有什么必经之路吗？本节将会从几个方面说明菜鸟和大牛的主要差距。

### 5.4.1 代码量的问题

农民伯伯比谁是种田能手的时候都是比收成，这是最有说服力的指标。IT 界的开发人员也是这样，代码量也算是开发人员的收成之一了，在一定程度上也说明了一个开发人员的经验和技能水平。因此在通往技术大牛的路上，代码量也是一个必不可少的衡量目标。

“师兄啊，我们谈了很多菜鸟成长的问题，那么菜鸟和大牛之间的差别到底在哪里呢？”

“好吧，今天我们就来说说菜鸟和大牛之间的距离。我先问你个问题，你从学习编程到现在的代码量有多少了？”

“这个嘛，我还没有算过，代码量是不是成为高手首先要完成的指标啊？”

“这是必需的，下面我们就来看看不同的代码量对应的不同发展阶段。”

#### 1. 1500 行~1 万行——天高任我飞

“首先，我们把代码量的起始点定在 1500 行以上，因为如果只有几百行代码的光辉历史是不足以与开发人员沾边的。”

“是啊，几百行的代码写个完整的五子棋都不一定够呢。”

“从这个起点出发，到 1 万行以内，这个阶段可以用‘天高任我飞’来概括，处于这个阶段的时候多半还在学校，对于软件开发的一些内幕并没有一个很深刻透彻的了解。因此对于自己洋洋洒洒几千行的代码很有成就感，认为已经灵活驾驭了一门编程语言，什么都可以编出来。”

“呵呵，很像我在大学的时候呢。写了个坦克大战就以为自己了不得了。”

身为尚未入行的准菜鸟，有这种想法也很正常，毕竟做学问是个由浅入深的过程，在初期难免会由于对行业认识不全而产生一些轻浮傲慢的想法。这种想法从一定程度上也增强了初学者的自信心，促使其再接再厉，向着下一个阶段进发。

就像故事中说的，这个阶段最好在学校中度过，而且是越早越好，因为初入职场的时候，最合适的定位是在第二个和第三个阶段之间。

## 2. 3 万行左右——我还很笨

“‘天高任我飞’这个阶段很正常，我在初学 Java 的时候也是经常为自己小成果窃喜不已，不过这些小成果现在看来的是很幼稚。”

“师兄，那经过过了‘天高任我飞’的阶段之后呢？”

“往往有人跟你讲道理，不管讲的有多正确，也不会完全信服，直到自己切身体会到了，才会明白。随着代码量的增长，达到 3 万行的时候，随着接触的技术和知识越来越多，慢慢地也就发觉自己之前的不自量力了。”

“是啊，知道的越多，未知的也就越多。”

代码量从 1500 行成长到 3 万行，不仅是技术能力和经验得到了很大的提高和丰富，对于整个行业和自己其中的位置也有了比较正确的认知。一开始是觉得自己什么都能做，到这个阶段是觉得自己还真有很多做不了的事情。就会自然地产生“寄蜉蝣于天地，渺沧海之一粟”的感慨了。

代码量达到 3 万行这个阶段越早到来越好，因为这个时候一个人的心态和工作模式已经基本成熟，唯一要注意的就是不要灰心，而要迎难而上，继续攀登。

## 3. 10 万行左右——登堂入室

“经过了 3 万行的阶段，算是进入到软件开发的正轨了。之后随着编程经历的丰富，当代码达到 10 万行左右的时候，算是个登堂入室的阶段了。”

“10 万行啊！那该是多么漫长的征途啊！”

“我可以这么跟你说，只要用心去努力，比较优秀的人在一年半到两年之间就可以达到这种高度的，就算是比较中等的人，三四年内达到这个阶段也是可行的。所以对于不同的人，这个指标的完成日期也是不固定的。”

“那登堂入室这个阶段是个什么境界呢？”

“一个人登堂入室之后，技术能力已经达到了较高水平，可以被初步称为高手，这个阶段应对一些中小型应用是不成问题的。”

代码量达到 10 万行标志着一个开发人员的初步成熟。作为一个立志在软件开发行业创出一番天地的读者朋友来说，这个指标还是非常有必要去完成的。虽然完成这个指标所用的时间因人而异，但最好不要让其超过三年。

当代码量达到 10 万行时，在自己从事的技术领域，已经勉强可以独当一面了，而想要成为真正的专家牛人，所缺少的能力应该是对于软件项目整体上的把握和统筹了。

## 4. 100 万行——万剑归宗

“最后，我们来说说代码量达到 100 万行的境界。”

“100 万行啊，那可得登堂入室十次才行啊！”

“100 万行算是软件开发人员的极限了。达到 100 万行的阶段叫做‘万剑归宗’，这在武侠小说中算是剑术的最高境界了。达到这个阶段，代码量算是完成了它衡量价值的使命，因为再提升代码量已经毫无意义，就像营养过剩后再摄入也不会有用。”


“那到了这个阶段应该研究些什么呢？”

“这个时候需要研究的，应该是类似于内功心法之类的了，比如专注于新技术方向拓展性的探索，或者是进行面向服务的底层平台的研发等。”

代码量达到 100 万行所需要的时间就更难估计了，这个指标只是编写代码的终极指标，能够达到这个目标的人不会很多，但是并不妨碍其进行那些“万剑归宗”的修炼。

这个时候一般不会再做具体的客户项目，更多的会去研究平台或中间件，力求能在领域内有所创新。比如进行下一代 Java EE 中间件的研发、下一代 SOA 平台的研发等。进行这种研发对项目经验有相当高的要求，否则开发出来的平台可用性将会很差，这也是为什么代码量需要达到一定标准的原因。

对代码量的衡量并不是局限于其本身，还包括一个人在开发过程中不断积累的编程经验和领域知识。编写代码只是修炼的一种形式，就像学习剑法，首先手中要有剑，然后随着内功的深厚，渐渐达到了“人剑合一”，最终达到“无剑胜有剑”的境界。

 **提示** 我们这里说的代码量是指有效代码量，如果每天只是敲“`System.out.println("Hello World!!!");`”之类的代码就算 1000 万行也是没有价值的。

### 5.4.2 敢于往上走一步

从菜鸟变成大牛，仅仅代码量到达一定数量还是不够的，往往还要取决于是否敢于多走一步，研究更深层次的东西。比如容器的技术原理、框架的内部工作机制等，这些往往才是高手和新手的最大差别。努力去钻研底层的知识，对于一个开发人员的成长是很有裨益的。

“蔡佳娃，刚才我们谈了代码量的问题，下面我再说另外一个大牛区别于菜鸟的特征。”

“好啊。那是什么呢？”

“学习使用 Java 这么多年，你应该对 Java 的开源框架和应用服务器了解不少吧？”

“呵呵，还行，框架像 Struts、Hibernate、JSF 啦，应用服务器像是 Tomcat 和 GlassFish 等，多少还是懂一些，能够应用的。”

“OK，那我问问你，曾经研究过某个框架或应用服务器的底层实现机制吗？这些都是开源的，细细品读过它们的源代码吗？”

“啊？这么高深的东西，我哪敢去尝试啊！”

“是吗？不过这正是真正的高手所做的啊！”

从事 Java 开发，肯定会用到不少的开源框架或是免费的应用服务器，很多菜鸟对于这些框架和应用服务器都是熟练运用即可，根本没有想过要深入的研究。就像幼儿园里的小朋友玩玩具，有的小朋友只是单纯地傻玩，有的则会搞破坏般地将玩具拆开来研究结构原理。所有人都知道，两种玩法的效果绝对是不一样的。

其实，除了上面提到的开源框架和免费容器，算法设计的思想、设计模式的理论、数学模型的应用等也是醉心于表面技术的开发人员容易忽略的内功心法。这些内功心法的特点是短期内不容易收到太大的回报，但是对于一个菜鸟成长为高手却是相当有用的。

“师兄，我就不明白了，你说高手都会去研究这些框架和服务器等底层实现和工作原理，可是我怎么就没有意识到这个问题呢？研究这些的作用有多大啊？”

“你就是太习惯用碗吃饭了，而没有想过研究碗是怎么做出来的。研究这些东西的意义可大了，你想想看，其实功能强大的框架和稳定的应用服务器不就是一个开发成功的软件项目吗？这不是正好可以研究一个成熟且成功的项目了吗？”

“说得也是啊，我以前从来没这么想过啊。”

“所以说通过看源代码研究这些东西的工作机制能让自己体会到这些优秀程序的设计思路 and 模式，对于自己以后的开发是很有帮助的。还有啊，很多时候基于这些东西进行开发的时候，会出现很多难以解释的问题，实际上问题的根源却是在这些框架或容器中。”

“这么说这些东西也会出错喽？”

“是软件就会有漏洞或 bug，我记得我当年学习 JSP 的时候，在用 Tomcat 开发一个网站的出错处理模块的时候，出错页总是无法显示。我当时找了许久，也到网上问了许多，都找不到答案，等我毕业进公司后，在有一次阅读 Tomcat 源代码的时候我才发现原来当年的错误是由于 Tomcat 的一个 bug 导致的，即当出错页面回显的内容长度太小的时候，Tomcat 服务器会忽略它不予显示。”

上面的故事以开源框架和应用服务器为例，向读者讲述了钻研这些高深知识的重要性。《荀子·劝学》中有云：“不登高山，不知天之高也；不临深溪，不知地之厚也”。完全不知道是一回事，研究过后就算不完全知道那也比没有研究过的完全不知道有价值多了。

敢于钻研这些高深技术的好处有如下几个方面。

- 帮助深入理解

比如通过阅读源代码对 Struts 框架的工作原理有了更透彻的理解，比起只会用标签的人来说，使用起来自然是更加清晰明白。

- 提高设计开发能力

通过研究设计模式和数学模型等抽象思想，可以在软件项目的规划设计方面有更高的认识，从而开发出功能更强，扩展性、可靠性更好的软件。

- 促进运用

就像故事中说的，对框架或应用服务器的底层实现有了一定的了解，可以发现其存在的一些特性或弊端，有助于更好地应用。

### 5.4.3 升天不成，掉下来也是个半仙

诚然，钻研这些高深的知识，肯定需要花费更多的时间和精力。如果彻底将其研究透彻，固然在能力和修为上会上升到一个新的境界。即便是最终没有完全弄明白，无法完全掌控好如此深奥的内容，当你从升天般的苦苦研究中退出时，你会发现以你的能力，你已经是一个半仙了。

“师兄啊，我发现这些东西很多都是超级深奥的。我觉得一时半会很难搞定，或者这辈子也只能研究个皮毛而已了。”

“呵呵，这倒不怕，你只要是用心去钻研了就好。彻底研究透是每个人都想尝试做到的，但就算你往返几次还是只弄了个一知半解，也不要灰心，它们是不会亏待你的。”

“事都没办成怎么还会不亏待呢？”



“你可以去试试啊！我给你讲一个例子你就明白了。我们公司的技术主管其实 Java 技术方面并不算是最牛的，而且他之前并不是搞 Java 的，而是研究算法设计思想的。结果也像你担忧的那样，他并没有在那个领域扬名立万，而是被那些更加优秀的脑袋挤了下来。”

“那他后来是怎么成为你们公司的技术主管的啊？”

“人家虽说是没有晋级成功，但是退下来往我们人群里一站，立刻就显出厉害了。结果很快就被提升到 Team Leader，随后又成为项目经理，以至于今天的技术主管。”

“哦，是这样啊，这让我想起了我的一个同学，他大学英语四级一直过不了，之前他一直在参加四级考前辅导班都收效甚微，后来发狠心一咬牙去报了个 GRE，学完之后回来看到四级考试题，结果在考场都笑出了声。”

在努力钻研这些高深知识的时候，不必抱着彻底拿下的决心和态度去学习。这些东西都是几代 IT 英雄智慧的结晶。如果被我们一眼洞穿其中的奥妙，那就太不切实际了。

就像余秋雨先生在《文化苦旅》中写到的那样，张大千举着油灯从莫高窟仅仅带走了一些线条，就让自己蜚声国际画坛。我们在钻研如艺术精华般的高深知识时，也不必要想着一口吃掉，哪怕只是领悟了其中的一小部分，后退一步，才发现真的是海阔天空。

## 5.5 酒香也怕巷子深

本书在前面的章节中曾不只一次地提到 IT 行业存在着激烈的竞争，如今在这个适者生存的世界，光靠自己的技术才能闯天涯已经远远不够了。酒香也怕巷子深，怕酒者的鼻子不好使，怕别人将自己的香味冲淡。

### 5.5.1 找到你的优势

在职场中，除了要明确自己的方向，还要找到自己的优势。每个人都有擅长和薄弱的环节，IT 菜鸟正处在事业的起点，认真分析和观察，找到自己在工作中表现强势的方面，扬长避短，才能让自己更具有竞争力。否则用自己的劣势和别人的优势去 PK，肯定会败得很惨。

“蔡佳娃，你现在刚刚步入职场，要刻意地提醒自己在最初的工作中找到自己的优势，这样才能让自己更好地成长哦。”

“找优势啊，这个问题很难哦。这么多年我都没发现自己有什么优势。”

“那是你没有用心去留意，是个人就会有与众不同的长处。如今你身在江湖，我想你应该不用我告诉你 IT 职场的竞争有多激烈了吧。”

“是啊，我们公司已经辞退掉一个跟我一起从试用期熬过来的人了，搞得我也是如临大敌一般，那师兄你说说该如何找到自己的优势呢？”

在职场中寻找自己的优势，最重要的还是那两个字：留心。刚刚进入职场，不要急着说我要怎样怎样，然后扛起锄头就下地，要多尝试不同的工作方面，在小事中和同事或别人比较，看看谁做得更好。这样一来，就会慢慢发现自己在哪一方面做得更为出色。

“还有啊，蔡佳娃，在寻找自己优势的时候，要注意发扬类似‘吃着碗里的，瞧着锅里的’这种精神才行。”

“那是什么精神啊？”

“是这样的。你在寻找自己优势的时候，首先是从自己正在从事的工作中尝试去寻找自己哪一方面做得尤为突出，这是面对当下的，算是‘碗里的’优势。比如公司目前主要面对移动平台的增值业务的开发，你就会在其中寻找自己擅长的方面，比如手机端套接字的开发等。”

“哦，那锅里的优势就是放眼不远的将来了？”

“正解，锅里的优势指的是未来的发展中哪些技能会让你如虎添翼，比方说未来你可以选择去从事企业资源管理项目的研发，也可以选择做电子商务平台的开发等。找到这些优势，自然会在以后的工作中慢慢向其靠拢，并最终将它们收入碗中。”

“嗯，师兄你说得太对了。我回去一定好好实践一下。”

故事中对于优势是一种按时间的划分，也可以按工作性质来将自己的优势划分如下。

- 技术方面

一般人都会在技术研发中找到自己的优势，技术中的优势还可以分为两种。一种是喜欢固定化项目的开发，即项目的开发技术比较成熟，需要注重的主要是面对客户不断变化的需求进行改进。另外一种喜欢研究新的技术，需要灵活快速地改变开发的模式和方案。两种方式并没有优劣之分，应该根据个人的优势来进行选择。

- 管理方面

有些开发人员在开发的过程中会发现自己管理方面的特长，管理也可以分为两种：技术管理和战略管理。技术管理仍然需要一定的技术功底，不过更偏重于管理方面，如项目经理、技术总监等。战略管理一般就是对公司的市场开拓、资本流动、技术方向、合作伙伴等方面的决策，如像李开复、唐骏那样的职业经理人。

在 IT 职场对于自己优势的发掘，很难仅仅通过坐在桌前拿一张纸列一个清单就能做到，需要在实战中不断地摸索出自身的优势才行。找到自己的优势后，才可以对症下药，让自己有的放矢，借着优势这股顺风扬帆远航。

## 5.5.2 学会竞争

身在职场，如何在明确自己的优势之后，在竞争中保全自己，在竞争中生存制胜，这些生存法都是菜鸟在成长为高手的过程中必须要了解的。否则不懂得竞争或是不恰当的竞争，都很有可能让自己在职场折戟沉沙，遭到流放。

### 1. 打好竞争这场仗

身在职场如战场，打好竞争这场仗，并且牢牢地守住阵地，才能让自己在乱世纷争中立于不败之地。但是打一场胜仗并不容易，需要准备的东西也就有很多了。

“师兄啊，以前老听你说 IT 职场竞争激烈，毕竟不在其中，我没有什么同感，现在自己在职场混了这么些日子，才发觉这里果然是个危机四伏的地方啊！”

“呵呵，没有这么激烈的竞争，哪来的咱们 IT 行业的时尚与多金啊！”

“我现在已经从求职这个战场生存下来了，师兄你再教教我如何在正常工作中正确地竞争吧。”

“首先呢，兵马未动，粮草先行。打仗先搞军备，准备工作你还是要做一下嘛。”

“那都需要做什么准备工作啊？”

“首先，打仗要有自己的主力部队，你的主力部队呢，就是你自己的优势。在战场上你的主力部队如果溃败了，那么你就该琢磨着是投降还是逃亡了。”

“是啊，要用主力部队去攻打敌人，这样胜算才会更大。”

“其次呢，打仗还需要谋略。不过 IT 这个战场就很精简了，你在职场只需要为自己经营一个很好的人脉就行了。”

寻找自己的优势这个话题已经在前面的小节中谈过了。其实不仅要找到作战的主力部队，会暴露出自己劣势的非主力部队也要想办法保护好，或者将其隐匿起来，不给对手可乘之机。或者加强训练，早日让其加入主力部队发挥作用。

人脉这个职场因素或许很多菜鸟并不是很了解，也并没有积累太多的人脉。人脉是一个无形的优势，发展人脉不仅仅是和贵人建立关系，还需要和形形色色的人建立联系。在务实的 IT 行业，人脉可能不会为你带来一场兵不血刃的胜利，但是至少会降低你遇伏遭袭的几率。

就像真正战场中粮草在“先行”之后还要源源不断地继续运送一样，在职场作战中的各种粮草还要一刻不停地进行补充。

## 2. 要良性，不要恶性竞争

良性竞争，就好比解放军训练中常进行的业务大比武活动，能激发每个参与者积极向上，刻苦训练，最终所有参与者的水平都能得到提高，是个多赢的局面。而恶性竞争中则经常会有一些鸡鸣狗盗的事情，同时也会产生很多不必要的损耗，最后导致的是群败的局面。

依据兵法和历代的军事武侠小说，恶性竞争在战场上、比武场上最为常见。但那都是指对外，而在 IT 职场，团队内部的恶性竞争则绝对是个害人害己的双头枪。

“蔡佳娃，我可要提醒你，在 IT 职场上和人竞争，无论如何都不要采用恶性竞争明白吗？”

“师兄放心，我肯定不使诈，不过，树欲静而风不止，万一恶性竞争冲着我来了怎么办啊？”

“有人对你放暗箭，首先要保证不要与其正面冲突，否则会暴露出更多的弱点。”

“那就是把牙一咬，忍了呗。”

“也不全是这个意思。当出现恶性竞争的时候，最重要的是避免内耗。”

“内耗是不是指和他进行周旋，拼个你死我活？”

“正解，内耗又耗时又耗力，耽误自己做正事不说，而且最后的结局必然是两败俱伤。不仅会让其他人当笑话旁观，最有可能出现‘鹬蚌相争，渔翁得利’的局面。”

“是啊，那样的话就是最大的悲剧了。”

“所以在遇到恶性竞争的时候，一定注意不要跟着耗下去，清自清，浊自浊。以退为进，在自保中让恶性竞争者不攻自破。”

古语有云：“木秀于林，风必摧之；行高于人，众必非之”。IT 行业再新兴，再前卫，也是由人组成的，必然会偶尔出现恶性竞争。下面给出了 IT 职场可能出现的几种恶性竞争场景。

### ● 场景一

某君敲开老总的办公室，神色凝重，忧心忡忡，老总忙问何事所困，某君欲言又止，最后开口说道：“我实在是不想说，不过，这关系到公司的发展……”然后某君宛如春秋战国时期纵横捭

阖的说客，充分站在公司的立场与老总推心置腹，研究某个人的发展前途。最后，聪明的老总应该弄明白了：哦，原来你是看×××不顺眼啊！

#### ● 场景二


某君与同事合作开发项目，联合测试的时候出了问题，某君不由分说就指责同事的模块有 bug，把错误责任推给同事负责的模块，还非要“真心真意”地帮助同事调试排错。

#### ● 场景三

某君在某一个小领域颇有建树，同事来请教该领域的问题，某君面露难色，知而不言，甚至在老总的压力之下也不愿与别人分享技术。有的时候就算是告诉别人了，也是“言而不尽”，只将解决问题的老笨方法告知，有时甚至是存在隐蔽 bug 的方法。

#### ● 场景四

某君在某一个小领域颇有建树，同事来请教问题，欣然应允，并坦诚相告。结果同事按此法去开发，整个项目出了很严重的错误。此时某君以救世主的身份火速到场，迅速将错误修正。蒙在鼓里的同事不知道自己已经做了一次绿叶来衬托“红花”。

 **提示** 以上这些可能出现的恶性竞争并不是存在于所有的公司，在开发人员技术级别比较高和公司企业文化比较团结正派的场合，这些恶性竞争鲜有所闻。而且明智的领导也是极其反感恶性竞争的，因为其不但不能使团队积极向上发展，甚至产生巨大内耗，影响团队的正常运转。而良性竞争则不同，它可以引发整个团队积极向上发展，最后参与竞争的人都能得到丰硕的回报。

作为一个正派的菜鸟，在遇到恶性竞争的时候首先要以自保为底线，让自己杜绝内耗。其次要以将恶性竞争者拉入良性竞争的队伍中来为最高奋斗目标，使自己的工作环境处于一种健康向上的氛围中，而绝不要积极参与恶性竞争。

再者，有了恶性竞争，难免就会在工作中出现不公平的现象。比如一个本该属于自己的培训机会、一个本该属于自己的空缺职位等。所以在这里还要劝慰各位读者在以后的工作中如果遇到了这种不公平，要学会有一颗平常心，尽量宽容地对待每一次不公平。

### 5.5.3 发展才是硬道理

谈论了这么多在职场面对竞争如何生存制胜的问题，令人感觉到职场确实犹如战场般残酷，又如江湖般险恶，真实的情况的确如此。不过对于这些棘手复杂的问题，有一个通用的最终解决方案——发展自己，因为发展才是硬道理。

“师兄啊，听你讲了这么多职场防守与反击的策略，我真的好头大啊。像我这种没心眼的人到了职场岂不是一千条命都不够死的啊？”

“哪有你说的那么严重啊！我只是提前给你打个预防针，未雨绸缪，让你提早做好防御工事，别到时候兵临城下了再悔之晚矣。”

“对对对，师兄你说得对。我现在就已经发现有人对我虎视眈眈了。”

“哦，没想到我的小师弟这么快就有了让别人嫉妒的资本了？哈哈！”

“哪啊，我们公司开发的项目一开始都是部署在 Tomcat 上面，后来有了一两个比较大的项目需要部署在 GlassFish 上面，而公司里就我还懂一些，所以老板让我去做，结果那个之前一直负责

项目的没风度的前辈就开始对我有些不满了。”

“呵呵，原来是这样啊！那我给你出个方法，这个方法是对付所有竞争问题的万能解决方案。那就是：拼命发展自己。”

发展才是硬道理，是解决一切问题的关键所在。在竞争激烈的IT职场，与其让自己花费时间去猜忌和怀疑对方是否对自己放了暗箭，不如自己努力提高，大步流星，远远地将其甩在身后，让他们仰着头伸长脖子张大嘴巴也无法望己之项背。

而且细细研究也不难发现，往往两个人之间的差距越小，竞争越为激烈。而对于比自己强大很多的同事倒很少嫉妒，更多的是崇拜，甚至立为奋斗的目标。因此，心无旁骛地让自己迅速发展壮大起来，超出那些恶性竞争者暗箭伤人的射程之外，不失为一种一举多得的生存制胜法则。


## 5.6 本章小结

菜鸟是一个让人自豪而又自励的名字，本章以菜鸟成长为高手的道路为线索，向各位读者朋友介绍了在刚刚步入职场时做事的学问，希望各位读者可以通过本章的学习了解到菜鸟在工作中应有的工作态度和方法，为自己即将步入的职场做好准备。最后，希望读者朋友能够做一个快乐并不断成长的菜鸟，最终成为大牛。



# 第 6 章 立足江湖——做人的学问

凭借着自己的知识和技术能力，相信每个努力过的人都可以在 IT 这个职场中找到自己的一席之地，但是想要更好地立足于这片江湖，就必须通晓一些做人的学问。但是很多开发人员都是与要做的项目关系比较亲密，两耳不闻窗外事，一片冰心在键盘。

 **提示** 如何为人处世是一门很值得研究的学问，光看市面上讲如何做人的书籍其畅销程度也可知一二。因为任何一个开发人员都不可能一辈子只和机器过招，在和上级、同事、下属、客户等不同的人群打交道的时候，都是需要一些处世哲学作为指导的。

## 6.1 新环境有新态度

要想了解做人的学问，首先要对自己所处的环境有个清楚的认识，到什么山唱什么歌，见什么方子抓什么药。就像工作态度一样，在认识你的新环境时，也要摆好自己的做人态度，这样才能在风云莫测的职场天地中有条不紊，游刃有余。

### 6.1.1 开发人员和厨师

对大部分开发人员来说，学校是职场的前一站，而 IT 职场和学校除了都把知识和技术奉为强盛之道外，并没有其他太多的相似之处。有些时候在学校的那些处世之道，拿到职场并不适合，有时甚至会逆行其道，起到相反的作用。

步入职场后，和牛开复师兄在一个城市工作的蔡佳娃，经常会和师兄一起吃饭，请教问题。

“师兄，进这行也有些日子了，我发现这里和学校还真是大不一样呢。”

“那是当然了，所以你不要把职场当做学校的升级版就行，很多在学校里行得通的办法到了这基本都没戏。”

“我觉得最大的区别就是在学校里能混，而在职场就没法混了。”

“哈哈，看来你在大学没少混啊！的确是这样，学校首先是教育你的地方，免费或是收取一定的费用；而职场是要掏钱给你让你干活的地方，单单这个区别，就决定了很多时候都不可以用对付学校的方式来应付职场。”

“是啊，两者本质差很多嘛。”

“举个例子来说，咱俩在这个饭馆吃饭。咱俩和后面的大厨师傅的心态就不一样。我们因为饿而吃饭，交了钱，却可以不吃，不吃也没人管。而大厨做饭就得留神了，饭菜做得不好就有可能拿不到钱。”

“是啊，职场的人就像大厨一样，而学校的人就像来吃饭的食客。一方是收钱提供服务，一方是交钱享受服务。”

故事中的例子也许不太恰当，但是也将学校和职场两个不同的生存环境做了一个基本的比较。

从学校跨入职场，需要转变的行为方式有很多，主要有如下两个方面。

- 莫见外

不要像羞赧的新生那样，等着有师兄师姐来把你“领上道”。要明白进入职场，基本不会有人来迎新，没人跟你客气，没时间也没人带你去观光熟悉。这是一个生存环境，不仅是生活环境。

- 莫等闲

不要像懵懂的新生那样，等着老师来督促学习，等着导师布置任务，等着和同学探讨交流。要主动一些，迅速投入战斗，马上做出业绩，这样才能很好地立足。

“蔡佳娃，我再给你讲个真事，我刚上班的时候，一起进来的一个人被开除了，那个人工作懒散不积极，属于那种好吃懒做型的，被开了也很正常，不正常的是第二天发生的事。”

“啊？第二天发生什么了？”

“第二天一大早，我们到公司，发现那个被开除的人的父亲领着他给领导赔礼道歉来了。看到这个你想到了什么？”

“这就好像在学校和人打架闯了祸，爸妈到学校跟老师校长说情一样嘛。”

“所以说嘛，一定要搞清楚自己在哪和自己的身份。职场可不是学校的延续，而是另外一种截然不同的社会场所。”

许多职场调查都发现了一个共同的问题：很多在职场受挫或烦恼的人，并不是因为技术能力、工作绩效不如别人，而是由于与上级、同事等的关系处理不好而备受煎熬。因此，学习研究IT职场的处世哲学就是一个不容忽视的问题了。

### 6.1.2 做人是为了做事

就像第5章讲到过的，在工作中要有菜鸟的态度，职场中做人也要有菜鸟的态度。菜鸟的做人目标就是让自己成为受欢迎的人，或者至少是不招大家讨厌的人，在为人处世中努力让自己的工作环境为自己的成长提供最大化的便利。

“那师兄你说说，像俺们这样的菜鸟，在咱们这个职场环境中该具备怎样的做人态度啊？”

“首先你要明确的一个道理就是，做人是为了更好地做事。职场中做人可以分为两个部分：第一个部分是如何做自己，就是应该把自己变成什么样的人；第二个部分就是与人交际，即处理好与每个有工作关系的人之间的关系。”

“总体来说，做人就是不断地和人打交道喽。那师兄你先讲讲如何做自己吧。”

“首先呢，我来消除一个你可能有的误区，那就是尽管我跟你说了这么多的江湖险恶，你可能也从网上看了许多职场的尔虞我诈。但是，IT职场的中流砥柱还是对技术的崇拜，整体的氛围也是大家努力提高自己，为公司和自己赚取利益。”

“是啊，不过多少也是会有一些明争暗斗和勾心斗角的吧？”

“那是难免的了，有人的地方就会有是非，针对这种很少出现的情况，也要采取正确的方法，要不然卷到这种漩涡里就很不好受了。”

对于这种小概率事件，职场菜鸟的中心思想是墨家学说所倡导的“兼爱非攻”，即以防为主，不率先使诈。身在职场中要以工作为中心，尽量让自己远离是非，如果被卷到这种漩涡之中，也

要尽快脱身，将自己置身事外，切不要试图与之周旋到底。

“师兄，既然做人主要分为做自己和处理与别人的关系。你先给我讲讲如何做自己，让自己成为一个受大家认可和自己满意的人吧。”

“OK，第一呢，你的人品值要达到一定的指标。”

“哦？我一直认为我的人品很好啊，呵呵。”

“首先，你要做个有诚信的人，这是立足的根本；然后是做个热情的人，对工作和对人都是；再做个助人为乐的人，帮助别人也就是帮助公司嘛……，反正你首先要人品好，中华民族的传统美德你至少得占到六七成。”

“嗯，这一点师兄你尽可以放心，我这个人绝对不坏。”

“好，下面我们就谈谈如何在职场中做好自己。”

下面主要列举几项菜鸟在工作中应当有的工作态度。

- 不结盟运动

在工作中，和大家的相处要融洽，但是作为一个员工，如果还不是领导，那么就让自己只代表自己。不结党，才不会有那么多的利益纷争，才能使工作环境更加单纯和平静。

- 不张扬

工作中尽量做到“两耳不闻窗外事，一心只读圣贤书”，就像应对恶性竞争一样，提升自己的修为永远是自己工作的重中之重。

- 公私分明

不要把私人的事情或东西牵扯到公司来，否则领导看了不舒服，又不好批评；同事看到老板没说什么就觉得你耍特权。

其实，做人的智慧最主要的还是处理人与人之间的关系，在 IT 职场中要面临的人群有很多，如何能来去自如地徜徉于人与人之间，绝对是每个身在职场的人应该重视的。本章后面的几个小节将会根据不同的人群为读者朋友提供相应的达人策略。

## 6.2 同事——战友和对手

同事，可以理解为共同做事，也可以理解为同时做事，前者为战友，后者就有些像对手了。的确，在职场中同事就是这两个相对对立角色的矛盾统一体。处理与同事关系的哲学，就是在这个矛盾中找到最完美的平衡点。

### 6.2.1 竞争与合作中的做人智慧

既然同事的身份如此特殊，那么在团队作战中，是应该同仇敌忾一致对外，还是暗藏玄机明争暗斗呢？真正的良性竞争是什么样子的呢？不妨用下面的一则小故事来说明一下项目开发中合作与竞争的存在形式和意义。

公司新接手两个分量差不多的项目，分别交给 A、B 两个项目开发团队，按照公司的规定，每个项目开发之后，都会根据项目的完成情况和开发人员的表现对项目团队的成员进行评比，表现好的开发人员会得到加薪或升职的机会。

A 团队每个成员都想获得加薪或升职的机会，于是每个人都努力把自己的那个模块做好以突出自己。同时，为了提高整个项目的质量，在可能的情况下，也会适当地帮助团队内部的同事。

B 团队每个成员更想加薪和升职。不过他们的策略是互拆后台，每个人都想背地里给其他人制造些麻烦，或是在团队分享中不积极，或是在调试上不配合，并打算以此来使自己的表现更加抢眼。

最后，当项目交工的时候，A 团队的项目由于团队成员的群策群力，良性竞争，完成得非常好，公司决定给他们集体加薪，并针对团队中表现尤为出的人单独加薪。

而 B 团队的项目由于团队成员各怀鬼胎，暗地拆台，导致项目延期交付，并且 bug 连连。公司不得不派 A 团队出来“善后”。结果 B 团队中的人哪还敢提加薪的事？不仅如此，他们还因为公司打算清除一些害群之马的消息而人心惶惶。

如果一个个拧干了放到秤上去称技术含量，估计 B 团队的人的能力不会比 A 团队的人差。不过就是对于竞争和合作这两个概念不够明白，做出了一些很不明智的事情才使得整个项目失败并因此波及所有团队成员。A 团队所表现的就是一种良性竞争。

到底合作和竞争是怎样的关系呢？可以从如下两个方面来探讨。

- 从严格意义上并不能说项目开发中竞争与合作是共存的，共存指的是宏观意义上的。很多时候在某一个时期，比如几天、几小时甚至几分钟的时间里，对于开发人员和其同事的关系来说，竞争与合作是二者之中取且仅取其一的。或者是在为了完善自己的工作而苦心孤诣、废寝忘食（竞争），或者是在为了整体项目的开发或调试而与同事协同奋战（合作）。
- 在项目开发中，合作是竞争的前提，没有合作就没有竞争。比如有一个项目任务，领导的目标很明确：团队成员通力合作进行开发，按时保质交付任务。这个目标也是每个开发人员的首要目标，然后在此基础上，开发人员也会有自己的目标，即想办法在工作中让自己有机会崭露头角。由此可以看出，如果没有第一个合作的目标，也不会有竞争存在的价值了。

了解了竞争与合作存在的意义，对于如何协调二者在工作中的比例以达到让自己和团队的利益获得双赢的结果，想必每位聪明的读者都已经成竹在胸。但是对于在职场中做人的智慧，恐怕仅仅靠竞争和合作的态度是不能够完全解决的，下面还给出了两个需要注意的方面。

- 弹簧的哲学
- 《易经》中的谦谦君子

### 1. 弹簧的哲学

把职场中自己和同事之间的关系比作弹簧，或许是再恰当不过的了。弹簧的两头是两个人，弹簧的长度就表示这两个同事之间的距离，人与人之间的距离远近在一定程度上也影响了关系的好坏。或许，好好研究弹簧的特性，会帮助我们更好地理解与同事的相处之道。

或许职场中最难以权衡取舍的是面对同事的求助吧，出于合作与竞争的博弈，在面对同事的求助时，总是有些踌躇不决。合理地处理好这个问题，基本上与同事的相处就成功了一大半。

下面将以弹簧的不同状态为例，探讨 IT 职场中出现的不同人际关系。

- 正常型同事关系

正常型同事关系如图 6-1 所示，两个同事之间的距离恰好是在弹簧的原始长度附近，此时二

者能够将关系维持在一个稳定的点上。而且因为距离适当，没有来自外界的力，双方都比较舒服。这种相处方式应该是最完美的了，两个人不给对方施加任何的外力，并在这种平衡下给予对方最大的协作，同时也保证自己不受外力的影响能努力工作。

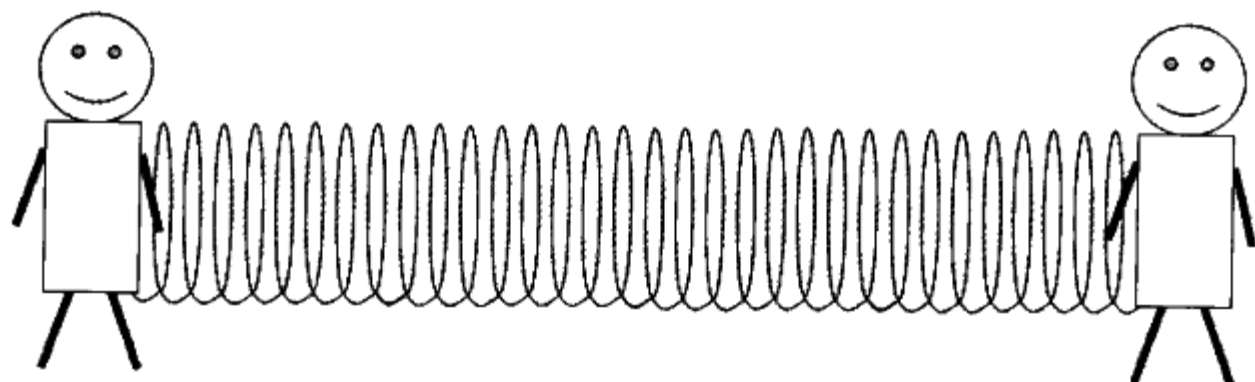


图 6-1 正常型同事关系

#### ● 压缩型同事关系

压缩型同事关系如图 6-2 所示，两个同事之间的距离拉得越近，弹簧被压缩得越厉害，在这种情况下，彼此之间受到的弹力也就越大，所以应该适当拉开距离，使双方满意。

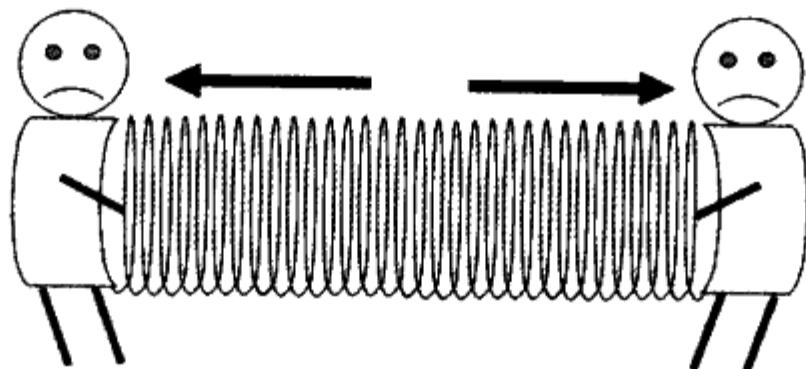


图 6-2 压缩型同事关系

这主要体现在同事之间的互相帮助中一种过分的助人，不管该不该帮的忙都帮了，结果不仅是被帮助的人失去了提高自己的机会，也使他更加依赖同事，如果有一天同事不再帮助他，他就会对那些同事产生嫉恨，彻底将同事关系搞僵。所以在这种关系模式下要学会适当地拒绝别人，顺应着弹簧的势能将二者的距离恢复到弹簧原长。

#### ● 拉伸型同事关系

拉伸型同事关系如图 6-3 所示，这种情况下两个同事之间过于漠视，距离已经大大超过了弹簧原长，会受到维持他们关系的弹簧往回的拉力。

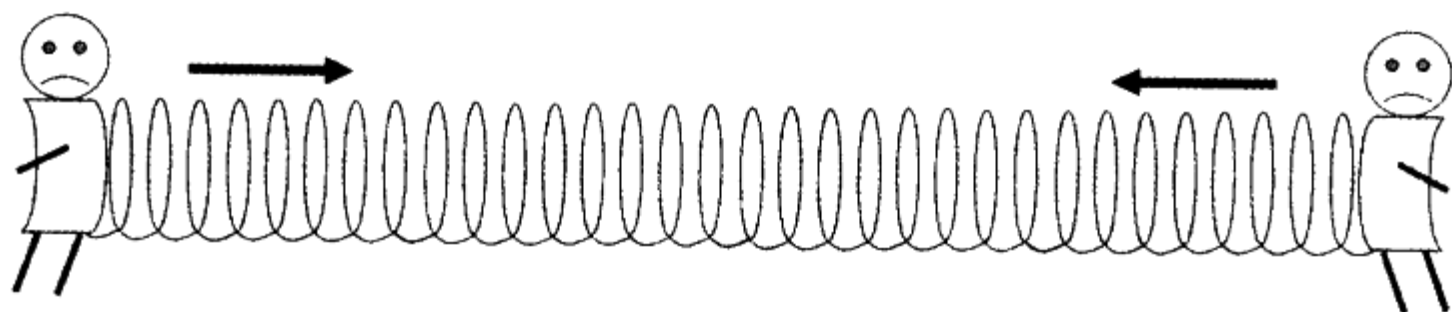


图 6-3 拉伸型同事关系

在工作中也是这样，同事之间的合作总是不可避免的，适当地帮助别人，不仅是在帮助公司创造效益，同时也为自己积累了人缘，切不可任何情况都据人于千里之外。所以这个时候就应该



顺着弹簧的势能学着拉近和同事的距离，一滴水只有放到大海里才不会干涸，一个人也只有放到更多的人当中才不会没落。

## 2. 《易经》中的谦谦君子

《易经》为百经之首，其中阐述的做人哲学可谓博大精深。《易经》中有一卦叫“谦卦”，讲述的是谦谦君子的智慧，谦谦君子，指的是既有高山般渊博的学识和突出的才能，又有大地般的忠厚宽容和虚怀若谷的谦虚之人。

从谦谦君子的浩瀚智慧中可以得出一个小小的结论，即做人的最高境界应该是“以强示弱”，对应到职场上的为人处世，就是能力很强，做出过很棒的成绩，但是仍然保持一种谦虚、柔和的态度。宽容地对待别人的错误，从不自吹自擂，唱高调。

与此同时，在“以强示弱”的做人态度之外，还有三种做人态度，这四种情况的分类基本上涵盖了所有身在职场的人可能达到的境界。

### ● 以弱示弱

这种人能力水平比较差，也没有什么大的业绩，但是能做到表里如一，对外也承认自己不够厉害，至少做到了谦谦君子的谦卑和自知。因此这类人虽然在职场难有大的作为，但是至少可以生存下去，有可能会靠着自已笨拙但坚持的努力让自己达到“以强示弱”的境界。

### ● 以弱示强

这种人在职场上就很危险了，明明自己能力不行，水平不够，偏偏要把自己吹得很高妙，五万的项目说成五十万，只负责开发界面说成负责核心代码。这样自吹自擂，别人听了不信则已，但凡有人信以为真，将重任交予此人，那不仅这个人会被人道毁灭，公司的利益肯定也要遭受巨大损失。

### ● 以强示强

这种人和谦谦君子已经比较接近了，不过他们缺少的是谦谦君子的柔和与谦虚。虽然自己就像自己说的那么厉害，一点不带假的，但是不加收敛的气焰也会得罪一些人，这种人在职场中会有不错的成绩，但是想往上走可能要困难一些。

## 6.2.2 做一个好同事

同事是每个人在职场中非常宝贵的人脉财富，同事一方面是上级提拔自己时需要考查和评估的群众态度，又是自己职位高升以后必然需要的基层支持。另外，若同事得到升迁也将是自己不可多得的一座靠山。所以，做其他人的一个好同事其重要作用不亚于做老板的一个好员工。

IT 职场中做人的智慧可谓深邃幽远，得之获益匪浅。不过人在职场漂，除了主线上的做人智慧之外，往往细节最容易奠定一个人对另一个人的态度基调，因此在与同事的相处中就需要注意一些细节，使自己尽量能够得到同事的认可和喜欢。

“师兄啊，听你讲《易经》，觉得我们的老祖宗实在是太有才了，我要是把这些智慧研究透了，那在职场中肯定是所向披靡啊！”

“呵呵，等你把 Java 技术研究透彻了，都不见得能彻底搞定那些为人处世的哲学。不过说了这么多深奥的理论，想必你就算听懂了也还是有些不知如何下手吧。接下来我们谈谈处理和同事之间关系的一些细节场景，看看你有没有犯过这样那样的错误。”

“嗯，我一定有则改之，无则加勉！”

### ● 场景一，不做炸雷型同事

早上，员工陆陆续续地走进公司。小李来得比较早，正在那里调试前天晚上没弄好的错误，正想着是不是应该在某个地方再加一条打印语句的时候，背后“啪”的一声挨了一重掌，差点没把早上的豆浆给吐出来。回头一看，原来是同事小王。

“李××，你猜我在路上碰见谁了？刘××啊，他辞职后居然在这边上研究生，我们都不知道啊。”小王的声音惹得左邻右舍都抬起头往这边观望。

“哦，是吗？那改天一起聚一下。”小李看了一眼小王，转头费力寻找自己十秒钟前被击飞的思路，而小王继续把这个雷人的消息告知每个被他逮到的同事。

炸雷型同事，就是不管走到哪里都像打雷一样炸开锅，而且杀伤半径至少是在街坊四邻之外，痛苦的是爆炸之后居然还有冲击波，需要老半天才能回归平静。

这种同事本身没有什么问题，心眼大多都不赖，只是做人太热情了，喜欢跟大家热闹一些。不过在 IT 的职场中，一个安静、不被人打扰的工作环境是相当重要的。因此，在与同事共事的时候要注意让自己轻轻地来，轻轻地走，不带走一片云彩。

### ● 场景二，不做指手画脚型同事

“咦，小李，你的这段代码应该放到 try/catch 中去吧？”

一回头，是老张站在身后，端着水杯，看来是去接水路过的。

“嗯，出这个异常因为你两边的网络协议不完善，这边已经不发了，那边还在等着收。”

还是老张，他怎么喝水喝得这么快？

“这样写不对吧，openConnection 函数还少个参数吧？”

又是老张，他没别的事可做了吗？

又过了一会……

“……咦，小李……你怎么关电脑了？”

身为一个开发人员，笔者也承认，在代码的编写过程中很多时候会出一些不算高级的错误，而旁观者的确也看得更清楚。我们并不是拒绝别人指出自己的错误，不过如果有人以此为生的话，那就很让人讨厌了，好像这些问题我们自己没办法解决一样。

不仅仅是针对错误的代码，或许每个开发人员都无法忍受一个并没有经过认真调查的人在那里对自己苦心编写的未完工的代码指指点点，也许这么做的人并没有恶意，不过这种守着别人犯错的作法的确不大符合大多数开发人员的心理。

所以不仅是代码方面的，工作中尽量不要干涉同事的工作，讨论方法、争论对错会有专门的场合，有时候人们宁愿让错误存活到自己发现或是开会时大家发现，也不愿看着你将它扼杀在摇篮之中，这就是人性，所以每个人在职场中都要注意给自己空间，也给别人空间。

### ● 场景三，不要习惯于揽下对工作无用的琐事

办公室有一个垃圾桶，可能是清洁工忽略了死角吧，垃圾总是没人去倒。一天，垃圾桶里的垃圾没有了，原来是小李倒的，大家心里对小李很感激。小李也觉得为大家做了一件好事，自

然心情也很舒畅。

随后，倒垃圾逐渐成为小李的日常活动，大家已经习惯让小李来负责垃圾桶的倾倒任务，小李虽然觉得很累，但是想想，这也算是为自己积累人缘，也就没再说什么。

过了两个月，小李生病了，三天没来。

大家突然很想念小李，想念他每天为大家做的日常任务，不过慢慢地，想念就变成了埋怨。

三天后小李来上班，发现大家对他已不像以前那般热情了，尽管他回来第一件事就是去倒垃圾。

后来清洁工开始每天负责倒那个桶里面的垃圾，小李再也没机会为大家服务了。

小李的初衷肯定是好的，做点好事，顺便也可以涨涨人气。不过最后大家对他另眼相看也不能全怨世态炎凉，大家已经把小李倒垃圾这件事当作了理所应当。哪天小李不在，当然会对他的“旷工”产生不满。

有的时候，刚刚进入职场的新手急于给自己创建一个完善的人际关系网，所以会做一些方便大家的事情来博得同事的好感。这种打通人脉的做法本无可非议，但是如果把做分外之事当作了一种习惯，让大家认为是一种应当应分的事情，不仅会影响到自己的正常工作，有时还会像场景中小李的遭遇那样受到相反的效果。

#### ● 场景四，哀多益寡的学问

“对了，蔡佳娃，上次你说你有次去部署 GlassFish 服务器，你们公司的一个前辈同事对于你撼动了他在项目部署方面的权威很是不满，还虎视眈眈的，后来怎么样了？”

“我哪有心思和他争啊，部署只能说是我的业余爱好，我在开发这一块忙得不可开交呢，所以后来再有这档子事我就全推掉闪一边了。”

“呵呵，不错嘛，你这个做人的道理正是我下面要讲的呢。当你的能力不算高时，对于你的主要工作方向，你应该尽力去争取，拼多狠也不算过。而对于其他方面，就算你很擅长，也尽量不要去操持，做好自己的本分是首要任务，这样也不容易得罪人。”

“嗯，师兄说得很有道理啊，看来我还有一些做人的天分呢。”

“而当你的能力很高的时候，就更应该懂得适当的放手了，如果能力高还要只手遮天，什么地方都有你，所有的功劳你都去抢，那些不如你的人还活不活啊！”

“是啊，学会舍得，才能更自在啊！”

职场中的生存，很多时候就像是在一个封闭的固定大小的空间里竞争，你的空间大了，别人的空间就会变小，就会压抑和不满。所以不管自己能力高还是低，都要给自己和他人留有一定的空间。

其实这个道理也是《易经》中一个谦谦君子的智慧，《易经》中谦卦有云：“君子以裒多益寡，称物平施”。意思是做人要注意平衡，多了就要减少一些，补充到不够的地方，减有余而补不足。对于那些能力强的人，拿东西别捧满，松开手指流出些来给其他人。

对自己不重要的东西就不要去极力争取，因为对你不重要的可能是别人所看重的。就像是故事中那样，如果蔡佳娃的那位同事仅有项目部署这一个强项，岂不要和蔡佳娃拼命！既然俗语说占小便宜吃大亏，那么在职场中不妨多放弃一些小便宜，让自己的工作环境更加舒心。

## 6.3 上级，不是校长或家长

虽然在职场中相处得最多的是同事，但是最需要重视的还是和上级的关系。说得通俗一些，和同事相处好会让一个人的工作环境变得更舒适，和上级相处好却有可能彻底改变自己的工作环境，让自己的职场更加好走一些。

### 6.3.1 是员工，不是学生

就像本章前面讲的很多人把公司当做学校一样，在职场中把领导当做校长看待的情况也有，这种人的心态还是不够成熟，总是把自己还当做学生来对待，觉得上级应该像对待校长那样来相处。这种人也是工作态度没弄清楚，没有认识到自己在职场中的位置。

“蔡佳娃，刚刚我们讲了如何与同事相处，下面来说说如何与上级相处吧！”

“呵呵，对啊对啊，同事只是战友，领导才是发工资的。”

“知道这个就好，既然领导是来发工资的，那就得拿出来准备领钱的样子，不要像个学生那样去工作就对了。”

“怎么算是像个学生啊？”

“比较典型的就是对工作不够重视，总觉得犯点错没什么，在学校不过是老师批评几句就算了，在公司犯了错误都是直接和经济利益挂钩的。下面我给你讲一个我同事的经历，你就知道如果真的是校长来领导我们是什么样的情景了。”

“哦，是什么样啊？”

“我的一个同事之前待的一家公司业绩平平，后来来了一个新的经理人，结果公司接连亏损三个月，每月损失二十多万。”

“啊，那是啥样的经理人啊？”

“是啊，那个同事也很纳闷，结果后来才发现，原来这位经理人干这行之前是个中学校长，碰到公司员工不好好干活，如给女朋友打电话之类的现象，他就叫到办公室谈话教导，结果整个公司的气氛懒散到不行，没几个真干活的，亏损也就在所难免了。”

从故事中可以看出，既然校长领导公司不行，那么以学生的姿态做员工也是很糟糕的。所以面对上级，不要凡事都随着性子来，要展现自己成熟的一面，让上级知道自己是完全能够胜任工作的。

残酷点说，老板就是把你当做赚钱机器来雇佣的，正像人们不会对出毛病的机器温柔一样，工作中尽量避免让自己懈怠。因为一旦犯错，并不会会有谈话、警告、记大过等绅士般的处罚，结果是或者将自己血汗钱的一部分奉上，或者拿了当月工资卷铺盖走人。

### 6.3.2 上级讨厌的员工

说到与上级的相处之道，肯定会有人冒出“伴君如伴虎”这句话。但是在 IT 职场，除了一些日韩企业和部分国企，很多情况下上级和下级之间的地位都是比较平等的，相处也较为随意，不会有那么森严的等级制度。

不过既然上级主宰着自己的生杀大权，在上级面前做人还是需要小心再小心，首先就是要避免做上级讨厌的员工，否则如果一开始就被上级列入了“黑名单”，那还如何再谈让上级将重任托付于己，更不用说在这个公司职途的光明与否了。

本书将 IT 职场中上级讨厌的员工简要罗列如下。

#### ● 沉默寡言型

公司在开会，主管王经理把项目团队的成员都叫到一起，研究到手的项目如何开展，大家各抒己见，都想把自己的想法融入到即将进行的项目中，场面很火爆。

王经理一边听大家的想法，一边为有这样一批热情奔放的手下而欣慰，见天色稍晚，决心要犒劳一下这群干劲十足的小伙子，于是便偷偷叫了外卖。

“来来来，这么晚了，我叫了外卖，大家先垫垫肚子再说。”

“哦，王哥你太敞亮了！”大家应和着，奔着香味而来。

“咦，王经理，怎么少一份啊，小刘还没有啊？”

“小刘？小刘也在吗？”

王经理是太高兴了，所以数人都没数对吗？不，是小刘太少言寡语了，王经理只是听声音凭感觉确定的人数，也不能全怪王经理，沉默寡言对于 IT 职场可是一个非常严重的扣分环节，这次只是误了一份外卖，下次是什么？奖金，还是升迁的机会？

现在，没人会把沉默寡言认为是为人谦虚低调、一心踏实工作，而且这种表现也会招来上级的不满。每一个 IT 公司要的都是一个可以出谋划策、有想法的员工，不是一个只听不说、深浅不知的员工。不管出于什么原因，沉默寡言，都会严重地影响到自己的职途。

#### ● 无主见不挑剔型

“小刘，公司准备派你和市场部的小郭去趟北京的客户那里排除一下故障，你看行不行？”

“嗯，我怎么着都行。”

“小刘，这个项目的我打算用 AJAX 或者 Java FX 开发界面，让你做你觉得哪个更可行？”

“嗯，我怎么着都行，您看着办吧。”

“小刘，公司准备开拓无线移动开发这个领域，刚接了个项目，你看能做么？”

“嗯，我怎么着都行，您看看有没有难度吧。”

“小刘，公司准备从你和小张之间选一位去上海参加项目管理培训，我推举了小张，你看行吗？”

“嗯，我怎么着都……啊？”

这种员工和前面那种沉默寡言的员工比起来好不到哪去，许多人在职场中都想让自己看起来像是花花草草那样好养活，随遇而安，这样显得不扎刺儿，不挑三拣四。

不过如果老板找你问话，说明他也有拿不准的事，那是对你的信任，这个时候最正确的做法是有一说一，据实以告，就算自己也拿不准，那也得像模像样地谈上几句。否则总是这样过分地随和，领导就会毫不留情地将这种没有主见的人打入冷宫。

#### ● 墨守成规型

“小刘，你开发的这个模块不行，回去赶紧改一下。”



“王经理，是什么地方不符合条件啊？”

“你的代码里面对于集合框架的操作怎么还是强制转换呢？”

“不强制转换怎么行啊，从集合框架里面拿出来不都是 Object 类型的吗？”

“我说你有没有了解过新技术啊？从 Java SE 5.0 开始就已经有泛型技术了，你们项目组的其他人都是使用泛型进行开发的。你的模块跟人家的有些地方对接不了，不改怎么行？”

职场中这种消极懒惰的员工是很难有所收获的，就像木桶能装多少水取决于最短的那块木板一样，一个领导肯定明白团队的战斗力很大程度上决定于能力最弱的那个员工。能力弱也就算了，还墨守成规，不知道与时俱进，那就更加招领导讨厌了。

长江后浪推前浪，前浪死在沙滩上，这就是 IT 行业的生存现状。所以要想不被汹涌而来的后辈挤出职场，不仅要在工作中做到与时俱进，学习新知识，还要不断地去尝试创新，用好点子、好想法让自己在职场的风口浪尖站稳脚跟。

#### ● 爱背后谈论别人型

“小刘，我让新来的那个小何写一份项目计划书，他对这里还不太熟悉，你有时间尽量帮帮他。”

“哦，王经理啊，我觉得这个小何没有他说的那么厉害啊，问的问题好多都很低级，而且他这个人做事很不专心，每天花好多时间上网聊天。”

“哦，这样啊。”

“倒是另外一个新来的小胡比较踏实，不过小胡的底子实在是差，听说是专接本考上去的。”

“啊，其实呢……”

“其实咱们公司里技术最好的应该是研发部门的副主管老许，不过这个人有个毛病，就是喜欢背后议论人。”

……

估计从这次谈话之后，王经理会对小刘倍加提防，他既然敢在自己面前对其他员工高谈阔论，保不齐下一刻就会在别人面前对自己发表“高见”了。这种人说的话虽然算不上危言耸听，但是由着这种议论传来传去，肯定会在公司内部产生裂痕的。

喜欢背后议论别人的人一般都有很强的负面情绪，这些人心中总是充满了不满与愤懑，觉得有人亏待了自己一样。任何一个明智的员工都会让自己主动远离这些“大喇叭”，不过就算只是单纯地喜欢讨论别人，那也是说者无心，听者有意，很容易影响到公司的团结。

“净坐常思自己过，闲谈莫论他人非”，不少领导也是从员工这个级别走过来的，也深知流言蜚语会像瘟疫一样破坏公司的士气和氛围，这种人肯定是上级深恶痛绝的。

#### ● 不肯奉献型

“小刘，晚上那个炼焦厂的客户要过来一下，说是有一些额外的功能需要详细谈谈，这个项目你比较熟悉，你来接待一下。”

“可是，王经理，我都跟您说好了，今天是我朋友生日，我要去帮他庆祝。”

“啊？你晚些去行吗，这批客户对公司很重要，所以要接待好。庆祝什么的晚一些、早一些都行。”

……

“咦？小张，小刘人呢？”

“他刚才走了，要不我去吧，这个项目我也有参与。”

“哦，哎，那也好，你跟他们协商好，我就指望你了。”

看似小刘拒绝留下接待客户的理由很充分，但是，毋庸置疑的是，从这件事后，在王经理的心中，小张的形象要远远好过小刘，尽管小刘的技术水平可能比小张高很多。

在上级的概念里，最好的员工必备的品质就是愿意为公司做一些适当的牺牲，适当地改变自己的个人计划以适应公司。一个忠诚乐于奉献、水平稍逊的员工远远比一个我行我素、坚持所谓“原则”的高水平员工具有更高的含金量。

#### ● 交际缺陷型

会议室里，客户正在和王经理商讨项目计划，小刘在一旁分发些材料。

突然，王经理的电话响了，王经理向小刘使个眼色，让他先和客户谈着，自己出去接个电话。

没等小刘表示什么，王经理就转头走出了会议室，剩下小刘面对着一群在他看来虎视眈眈的客户。

五分钟后，王经理回来，进门就说：“哎呀真不好意思，接了个电话，谈到哪了？”

“没谈，等您来呢！”这是小刘激动得有如见到亲人般的声音。

可以想象王经理此刻的表情和想法。他留下一个不会交际的小刘，让客户在他打电话的时候干等了五分钟，估计这个小刘将会被王经理直接拖入“黑名单”了。

随着职务的升高，需要打理的人际关系也会越来越复杂。因此一个交际有缺陷的人，除非是坚定不移地走技术路线走到底，否则很难顺利开展自己的职途。而上级需要看到的是一个能为自己独当一面的得力助手，如果无法满足上级的需要，自然也不会得到赏识。

#### ● 借口频频型

“小周，这个问题是怎么回事？为什么你的模块开发进度落后别人那么多？”

“他们迟迟没有定下通信协议，我拿不到协议只好干等着。”

“那你怎么不先进行界面的开发呢？”

“我想着界面到最后还不是要统一制定，就不如最后再开发界面。”

“怎么这么多理由，不管了，回去加快进度，后天必须拿给我成品！”

“可是，明天是端午节……”

“怎么？你连明天的借口都想好了？”

身为一个上级，没有什么比坐在椅子上看着下属为自己的过错找借口更烦恼的了。这种爱找借口的人要么光说不练，没有真才实学；要么好吃懒做，不认真工作。

老板要的不是理由，是成果。错误已经犯了，把下属叫来指出错误是为了让其改正错误不要再犯。听一堆借口一点用都没有。老板心目中的好员工，绝对是错了就干干净净承认，不为已发生的事找借口，只为能决定的事情而努力的员工。

#### ● 只手遮天型

“真是气死人，如果没有我，哪有公司的今天？进度催得那么紧，我稍微划划水有什么不好，

不都是为了公司的利益吗？至于要扣我奖金吗？”原来是公司的技术一号人物大刘，大刘刚刚被老板叫去，可能是挨骂了。

大刘越想越气不过，想想自己在公司这么些年的劳苦功高，随后决定让老板知道没有自己的公司将会是什么样。

第二天，大刘没来上班，说病了，请假半个月。这下公司可慌了，大刘在公司是经验最丰富的，对公司所有的产品都非常熟悉，技术方面更是有别人无法企及的水平。好几个正在开发的项目都是由大刘负责的，公司基本上是瘫痪了一天。

早有大刘的眼线回去报告，大刘心想总算出了口气。不过仍然拒绝上班，老板打电话也无动于衷。

三天之后，公司决定痛定思痛，不再奢求大刘回来，整个公司开展自救活动。大家断了请大刘回来的念头，只好自己咬牙坚持，渡过难关。

十天后，大刘火速复工。

大刘为什么要提前复工呢？心疼公司了？真实的情况是，大刘如果不复工，公司会慢慢地不再依赖他。这个称病要挟的计谋大刘的确是用错了。

很多人在公司做到一定高度后，就觉得自己只手遮天，公司离不开自己了。其实离开了谁公司都不会垮，如此夸大自己的位置，并以此威胁老板，只会让自己更加难堪，让老板更加讨厌自己。

一个技术骨干离开了自己的位置，立刻会有人补上去，虽然技术可能没那么好，但或许是因为长期被技术骨干压制着得不到发挥，假以时日，搞不好会因为技术骨干的引退而更加出色，到那个时候，就真的用不着技术骨干的回归了。

#### ● 越级活动型

“请进。”门开了，一个不太熟悉的人走了进来。

“王总，我是技术部的王××，我们部门正在进行一个项目的开发，我想到了一种方案，想来跟您说一下看行不行得通。”

“哦，你怎么不去找你们部门的胡经理？这些问题跟他谈就好了。”

“我觉得这个方案胡经理可能没办法决定，所以直接找您来了。”

……

在职场中，越级活动的原因很多，有的是因为上级赏罚不公，自己的回报比付出少；有的是不喜欢上司，直接打小报告；有的是不信任上级，害怕功劳被抢；有的则是眼高手低，觉得上级无法理解自己的想法，需要找个更加识货的人来。

抛开越级活动的种种动机，任何形式的越级活动都是对自己职途的自我毁灭。这么做如果没成功不了了之还好说，如果越级成功，自己获得一些提拔或重用的同时，也会得罪了自己以前的直接上司，以树敌的代价获得职途的成功，实在是不太划算。

三国时期的吕布为何白门楼殒命？因为大家都怕了，勇猛善战的悍将吕布总是带给人们“弑主”的新闻，哪里还有人敢留用？对于越级活动的人公司领导也会对其有所防备，因为今天是越过了别人到自己这里来告御状，搞不好明天就会越过自己跑到更高级的领导那里胡言乱语。

身为一个领导，都喜欢从手下中挑选精明能干的人来重点培养为自己的左右手，往往都会从小处着眼观察一个人。以上说的这些虽然都不是什么大毛病，但如果这些细节问题处理不好，遭到领导厌恶，那么职途的前景就会黯淡一些了。

### 6.3.3 怎样与上级处理好关系

只是研究了上级讨厌的员工还不够，这只是让自己免于被讨厌。要想成为上级的得力助手和重要心腹，还是要在日常的交际中积累上级对自己的信任和赏识，这主要包括如下几个方面。

- 面对不能选择的上级
- 功高盖主了怎么办
- 了解自己的职权范围
- 珍惜每一次与上级交流的机会
- 为公司创造价值才是硬道理

#### 1. 面对不能选择的上级

“师兄，你刚刚列举了上级讨厌的几种员工，那怎样让自己成为上级喜欢的员工呢？和上级打交道应该注意些什么呢？”

“让上级喜欢很简单啊，只要按照我刚才说的那样反过来做就行啦！呵呵，开个玩笑，其实光这么做还不够，很多时候还要面对一些其他的问题。”

“那是些什么问题啊？”

“比如说碰到你不喜欢的但是无法选择的老板怎么办？”

“唉，还别说，真的没有想过这个问题呢。上级还真不是自己选着跟的。”

“呵呵，一个优秀的上级就像是一个领导者，会把你变得像他那样出色。而如果被一个并不优秀的上级领导，那就是另一番天地了。如果不懂得处理和自己不喜欢上级的关系，那么就算自己显得不讨厌，也是没用的。”

读过《三国演义》的都知道，在这本书中出现最多的一句话大概就是“良禽择木而栖，贤臣择主而事”，意思是有才能的人应该选择识才的君主。基本上这话说完以后都会有一个难得的人才背叛旧主，另投高人去了，这就是在东汉末年比较火的炒老板。

不幸的是，在当今的IT职场，不管你的能力够不够得上难得，很多时候是不能随随便便炒老板的。既然无法改变现状，那就应该努力适应他，有时人与人之间的相处，就是看双方的妥协程度。而且任何一个人，能走到领导的位置，必定有其他人无法比拟的过人之处，技术好，或是人脉广，姑且将其学到手再说。

但是适应上级并不是盲目地追随，只是尽力确保自己能配合和执行上级的工作目标，不能以牺牲自己的本性为代价，换取与上级的和睦相处。

#### 2. 功高盖主了怎么办

工作中会遇到自己不喜欢的上级，同时也会遇到在某些方面比自己能力稍弱的上级，这个时候的处世之道就有不同了。

决定楚汉之争关键的韩信，算是个功高盖主的最好例子了。韩信在投奔刘邦之后，先是明修

栈道，暗度陈仓，取得对项羽的初胜，而后又背水一战，以不足十万兵力大胜二十万赵军，随后又攻破燕、齐等国，最终在垓下与刘邦合围项羽，迫使后者兵败乌江而自刎。

立下震主之功，名扬天下的韩信，却在大汉建立之后慢慢走向自己的末路，功高盖主的他使得汉高祖刘邦对此提心吊胆，最终将他除之而后快。

工作中有意或是无意间的功高盖主，虽不像大将军韩信那样会招来杀身之祸，不过如果处理得不好，也会影响到自己的职场。再英明的上级，也是需要领导的威严和架子的。

首先是不能炫耀，要有感恩的心态，一些欧美的公司或是豁达的上级是可以容忍的，但是一些日韩公司或是国内大部分公司的领导还是比较忌讳这种现象的。不仅要保持低调，还要不时地给领导面子，让领导确定对你的领导地位，让领导认为你是一只拴得住的野兽。

如果想以此邀功获得加薪和升职，也要采用正确的方法，不可以耍挟，也不可以向上级的上级告状，搞隔山打牛，否则就算得偿所愿以后在公司的日子也不会好过。如果实在是协商未果，假使对自己能力有信心，跳槽也不失为一种很好的解决方法。

### 3. 了解自己的职权范围

新入行的人，首先要明确自己的职权范围，什么能做什么不能做，老板喜欢的员工首先是守规矩、听话的员工。技术再好也不能超越自己的职权范围，管东管西，尤其是毫不留情地指出上级工作中的问题，和上级研究工作的改进问题等。有才能轻狂一些无可厚非，但是如果连自己该做什么不该做什么都不知道的话，将很难得到上级的赏识。

### 4. 珍惜每一次与上级交流的机会

每一次与上级的交流都是提高上级眼中自己形象的大好机会，所以一定要重视，要在事前做好充足的准备。尤其是在向上级提交一个自己方案的时候，一定要保证自己有完备的计划，这样即使上级不采用你的方案，也会对你的认真和敬业大加赞赏。

### 5. 为公司创造价值才是硬道理

说到底公司存在就是为了利益，所以不要在和上级的相处上花太多的心思，能够为公司挣来实实在在的财富，才是和上级融洽相处的前提。否则光说不练迟早会被揪出来清理出团队，新入职的人更应该明白，不断为公司创造价值才是获得领导赏识的最佳途径。

## 6.4 新人和下属，曾经的你

进入职场，不能总想着如何做个好菜鸟，做个好员工，随着技术的提高和经验的积累，自己总有一天也会有自己的下属，成为德高望重的公司顶梁柱。这个时候需要打理的人际关系就需要再加上那些新入行的菜鸟和自己的下属了。

### 6.4.1 准备工作

受到提拔成为了领导，很多人都会想着要报答老板的知遇之恩，感谢广大同事的鼎力支持。但是往往会有些急功近利，在对待下属的时候，也就会暴露出很多不正确的管理之道了。

“师兄啊，我遇到麻烦了！”



“哦，什么麻烦啊？”

“我们正在做一个项目，但是我们团队的 Team Leader 突然生病了请假三天，临走时他跟项目经理说让我暂时顶替他的空缺。我一下子从员工就变成了一个小领导，虽然只管着四个人，不过这也够我烦恼了，我该怎么和他们处好这三天的关系呢。”

“呵呵，原来是为这个烦恼啊？迟早要做领导的嘛，提前感受一下也不是坏事。不过虽然只是三天，也是不能小觑的。”

“是啊，那师兄你说我该如何度过这三天呢。”

“首先，既然你算是一个小领导了，就要有领导的样子，不可以还像从前那样。但是你不能架子太大，在技术至上的 IT 职场都是以才服人的。”

“哎，想不到这三天的领导也不好当啊。”

每一个受到提拔的人，首先要费些工夫研究的就是如何同曾经与自己竞争过这个职位的原同事以及其他同事搞好关系，这是一个非常考验人的难题。原先的同事做了自己的上级，看得再开心心里也会有些疙瘩。所以重新收获旧同事的支持，将会是一笔不小的财富。

从一个开发人员升任至项目或部门经理，工作重心也会发生一些变化。原先会想着如何让自己在团队协作中为公司多创造财富，在公平竞争中崭露头角。而现在需要考虑的，就是统筹来自各方面的人才来为共同的项目努力，也就是做员工时执行多于计划，而做领导后计划多于执行。

作为领导，最需要考虑的就是想着如何赢得下属的尊重，带领自己的团队开拓进取，扶持有前途的下属。那么，走马上任后，就必须做出一些改变了。

- 对于自己的升迁，不要刻意去解释和证明什么。无谓的解释只会增加他们的怀疑和作为失败的竞争者的不甘，而急于证明自己的实力也会在自己和下属之间产生敌对的想法。所以最好的方式就是不予解释，不需要对任何人有愧疚。
- 自己还是原来的自己，只是职位变了，能力和经验都和从前一样。要认识到下属才是自己最主要的战斗力，所以一定不可以升职就不再把下属当回事。但是领导的态度还是要有的，否则失去威信就很难驾驭手下的人才了。
- 多想想自己以前做员工的时候对理想上级的构想，避免自己得意忘形，变成不懂得尊重下属、不考虑下属感受的讨厌上级。

#### 6.4.2 学着做个好领导

要是管理下属真那么简单，社会上就不会有这么多商学院了，IT 职场上也就不会有这么少的像唐骏、李开复这样的打工皇帝了。好领导首先是个好员工，像李开复也在自己的专业语音识别技术领域取得了非常突出的成就，其次就需要一些为人处世的哲学来使自己带领的下属为公司创造更多的利益。

做一个好领导的管理哲学有很多，本书只是根据 IT 职场的现状，将一些比较常见的上级正确对待下属的方式列出以供参考，主要包括如下几个方面。

- 给下属明确的信息
- 相信下属
- 不吝表扬

- 敢于承担责任
- 不怕被超越
- 广开言路

### 1. 给下属明确的信息

作为一名下属，肯定都有过苦于无法理解上级工作指示的烦恼，所以在自己做了领导之后，就要避免让这种困惑再次发生在自己的下属身上。下属做得不好很多时候都是没有彻底理解工作目标与重点，所以当一个好上级的首要任务就是为自己的下属明确工作的方向。

### 2. 相信下属

身为技术人员，有的人就算走上了领导职位，也还是放心不下，担心下属的技术水平有缺陷，害怕会出问题影响自己没坐稳的位置。有的甚至仍然身先士卒，事必躬亲，这样不但使下属没机会亲自下手，自己还会把领导的正常工作如统筹计划、团队协调与管理、联络客户等给耽误了。

三国时期的奸雄曹操能统一北方，势压蜀吴，就是因为他可以做到“用人不疑，疑人不用”，虽然因为多疑错杀了好多忠臣良将，但是曹操对手下的信任还是招揽了一批忠肝义胆的人才，使他可以多次险处逢生，被手下所救。

因此作为一名领导，就应该相信下属，放心大胆地把工作交给他们，一个领导的工作，应该是不断的发现和重用人才。因为他们的成功，也是你领导的成功。

### 3. 不吝表扬

作为一个上级，激励下属的最好方式就是奖励，有时甚至不需要什么物质上的奖励，上级口头上的表扬对于一个员工也是非常受用的。

楚汉之争，项羽落败，除了他的自负之外，很大一个原因就是项羽对于手下的吝啬。作为将领，他待人温和慈爱，与士兵同甘共苦，共同进退。但到了攻破城池论功行赏的时候，就会“印刖敝，忍不能予”，把授权的帅印摸了又摸，棱角都快磨没了，就是不肯分赏给众人。手下的人拼命却得不到回报，忠心也就慢慢磨没了。再加上他的刚愎自用，逼得很多谋士武将都转而投奔刘邦去了。

打仗是这样，在职场中也是这样，想要留住人才，就要不吝奖赏，让他们在自己手下更加努力的工作。不过要赏罚分明，要注意指出下属工作中错误的地方，不可以纵容、姑息手下人犯的错误。

### 4. 敢于承担责任

领导与普通员工最大的区别就是要承担责任，当一个项目做得成功的时候，把功劳都分给下属，跟他们说他们做得很好；项目失败的时候则需要把失败揽给自己，勇于承认自己的失误，这种能扛起责任的上级才是下属敬佩的对象，才能够让下属跟随效忠。

### 5. 不怕被超越

前面也已经提到过，做一个领导的重要任务就是培养和发展下属，不限制他们的飞翔，如果

自己的手下变得比自己更强，应该欣慰才对，千万不能搞武大郎开店。

篮球明星乔丹之所以能够成为很多人无法超越的“神”，很大程度上就是因为其卓越的领导才能，在他率领公牛夺冠的征途中，乔丹并不是全队得分一直最高的人，很多时候队友的得分都在他之上，乔丹对此非常开心，他不怕被队友超越，反而会不断地鼓励队友得分。

## 6. 广开言路

“将拒谏则英雄散，策不从则谋士叛”，面对来自下属的 idea，每个领导都应该耐心倾听。听一个想法也许只会耽误 3 分钟，但如果是个好点子，那么这 3 分钟就非常值了。不管下属的想法自己需不需要，听听都无妨。如果轻易地拒绝听一个不好的点子，可能也就拒绝了随之而来的其他好点子。

### 6.4.3 被夹在自己的上级和下级之间怎么办

还是个员工的时候，已经是底层的职位了，只需要向上和领导打好交道即可。但是做了领导之后，在向下处理好自己与下属关系的同时，也要向上兼顾自己的上级。作为老板和员工之间的联系中转站，如何传达老板的指示和反馈员工的信息，就显得非常重要了。

“师兄啊，这两天的替补领导可把我难受坏了。今天老板叫我过去，说项目进度太慢，要我对我们那个小组严厉些，督促他们加快进度。我还没说呢，我该不该说啊？”

“卡在自己的上级和下级之间的确是不太好做，领导的话你传还是不传很值得研究。如果老板对你催得很急，你回去也对团队里的人催得很急，那么有可能团队里原本设想好的解决方案会没有时间采用，仓促之间或许会偷工减料难以保证项目的质量。”

“是啊，那我就什么也不说了？这样好像也不行啊！”

“你们团队的项目，你肯定也十分清楚，领导说要加快进度，那么在你看来是你们比较懒散呢，还是这个项目的规划有问题啊？”

“这个问题倒是没有，只是这帮客户总是要添加新的功能，总是在变，所以前期走了不少回头路。导致现在进度有点拖后腿。”

“好啊，既然是这样，你不妨对老板如实相告，告诉他如果催得太紧，那么只好在质量上稍微委屈一下，偷工减料或是敷衍了事。”

“啊，这样可以吗？这不是威胁老板了吗？”

“一个明智的老板，一定会分清是不是威胁的，肯定会知道该怎么做的。”

巧妙地协调好老板和下属之间的矛盾，是每个中层领导都要重视和研究的，软件开发需要一个宽松的环境，是一个创造性的工作，不可以有过多的压力。如果仅仅做了老板的传声器，领导给了自己一颗炮弹，自己回头也往下属堆里面扔一颗，这样的上级是无法为公司创造财富的。

如果更有甚者，做了领导的放大器，领导扔一颗，自己给下属投了十颗炮弹，那么这种上级就是 IT 职场中的恐怖分子了。正确的做法是老板给自己扔一颗炮弹，自己回下属那里打个喷嚏足矣。这样将上级的压力过滤在自己这里，让下面有一个比较宽松的开发环境，使自己的手下都可以心无旁骛地发挥自己的聪明才智，快速创造财富。

本书在第 1 章曾经介绍过微软项目开发的团队模式（参见图 1-2）。结合那张团队模式图，我们来研究一下微软的项目角色分割，如图 6-4 所示。

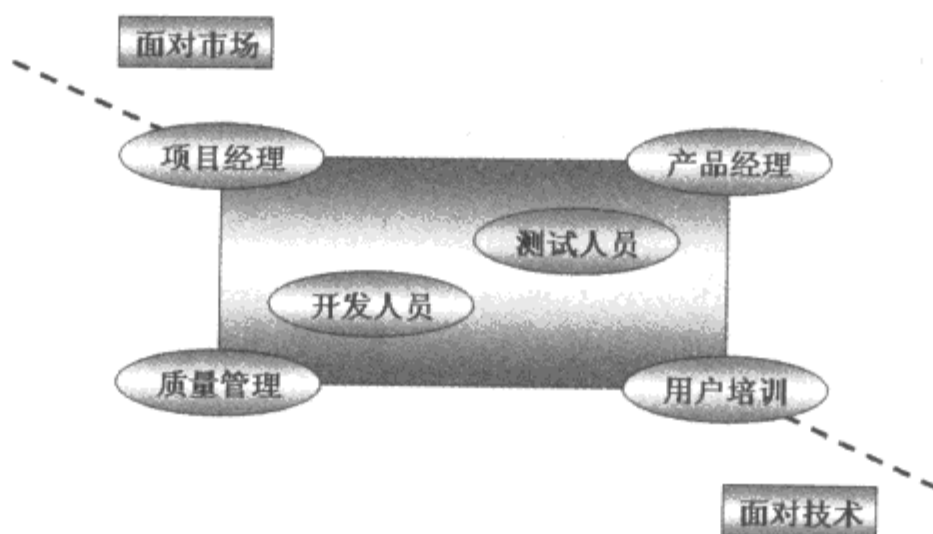


图 6-4 微软项目角色分割图

在图 6-4 中，项目的开发人员和测试人员被其他的团队角色包围在一个密闭的环境中，只和团队内部的其他角色交流。开发人员只需要按照项目经理的指示进行功能的开发，而测试人员只需要对开发人员开发出的成果进行测试和反馈，他们都不会感觉到外界施加给他们的要求和监督。

而与外界进行交流的工作交给团队中的其他角色，如产品经理负责和客户协商项目的功能实现，主要面对市场，而质量管理则负责技术方面的可用性和易用性的评估。项目经理和用户培训这两个角色，则既负责技术上的业务又负责市场方面的业务。

这种团队角色分割方法既保证了开发人员的效率最大化，又使得团队与外界客户保持及时的沟通，避免了开发人员直接承受压力导致技术掺水的现象发生。

## 6.5 客户，领导内行的外行上帝

前面已经介绍过了，软件开发行业是服务业，服务业没有市场和客户是万万不行的，所以要想在 IT 职场走出一片天地，必须学会处理和客户之间的关系。正如网上经常会看到的一句话：“小生意看做事，大生意看做人”。

客户虽然只是外行，关心和擅长的也不是技术，但是身为内行人的 IT 开发人员却必须满足这个外行上帝的需求，很多时候都是技术执行得很成功但是和客户的沟通不够或是交流不理想，给项目的开发和实施带来了很多不利的因素。

### 6.5.1 如何招待上帝

既然经营一个良好的客户关系是每个 IT 职场人尤其是想要做到高层人的必修课，那么在开课之前，首先需要了解一下 IT 行业都将面临什么样的客户。

“哎呀，师兄，我总算是熬出来了，我们团队的 Team Leader 终于回来了，这几天可把我折腾坏了。从来没有这么想念过他。”

“呵呵，才三天就这样啊，不是也没出什么大乱子嘛。要这样等你和客户打交道的时候不更得应付不来了啊！”

“这三天还全靠师兄呢。不过和客户打交道应该容易些吧，我上次给客户做个项目介绍好像也没那么困难啊！”

“那只能算是打交道中很小的一方面，和客户打交道的学问也大着呢。和同事搞不好关系也只能是得罪一个人，如果和客户没有处理好关系那就是没钱吃饭的悲剧了。”

“哦，说得也是啊，那该如何招呼好这些掏腰包的人呢？”

“首先你要研究好你所面对客户的类型。”

这里的客户类型并不是按客户的行业领域如银行、金融、电信等进行划分的，而是根据客户与所开发项目的关系来区别的。按照这种分法，客户可以分为以下三种。

#### ● 项目目标部门领导

这类客户是决定要进行软件项目开发的人，身为某个部门的领导，或者是要做出业绩，或者是要为单位创造效益，决定依靠软件技术改良本部门的工作流程以提高效率。这类客户主要出现在项目开发一开始的协商阶段和最后的结款阶段，主要关注的是该软件项目是否能够给自己的部门带来利益、投资回报比高不高等。

#### ● 项目目标部门的员工——软件的直接使用者

这类客户是前面一类客户的下属，是所开发的软件项目最终的使用者，也是主要需要打交道的一类客户。软件项目的使用者不一定是软件项目的支持者，因为有可能新生产方式的引入会对其原来的工作模式造成影响。这类客户出现在项目开始之后，一直到项目最终交付。

#### ● 项目目标部门的上下级或兄弟部门的人员

这类客户虽然不直接使用软件项目，但是要求开发出来的软件系统能够和自己的信息系统无缝结合，而且运行效率和可靠性只能比原来的旧系统高。如负责开发某个公司的财务软件系统，必须保证该公司销售部门的信息系统能够将每日的销售金额更新到新开发的财务软件系统中，并保证兼容性。这类客户一般和软件的使用者同时出现。

“师兄，听了你这一番话，原来与客户处理好关系还是蛮费劲的嘛。”

“那是当然，面对的客户不同，所要做的工作也不一样呢。”

“那具体都采用什么策略啊？”

“面对第一种客户，即发起软件项目的人，这些人都是领导，关心的最多的还是公司的效益和自己能否通过这个项目的开发创造业绩。所以在与其协商的时候要把我们的软件系统的优势说出来，要让他们相信我们的软件系统是可以为他们公司带来效益的。”

“哦。那对于第二种客户呢？”

“要注意到第二种客户中也有对项目持怀疑和不同意态度的人，因为可能新的软件项目会威胁到自己原来的工作或者带来一些不愿进行的工作方式的改变。所以和第二种客户打交道主要就是针对这些情况，让他们相信我们的软件系统一定会提高他们的生产效率，而且易用性也很好。”

“嗯，第三种客户就比较简单了吧。”

“相比来说是这样，对于第三种客户我们只要让他们看到我们的系统至少不会减慢和影响到他们本身的工作效率就行。”

同时，在软件项目开发的不同阶段，和客户打交道时所要做的也是不尽相同的。根据不



同的项目阶段与客户进行交流内容的不同，可以把一个项目的生命周期分为如下几个阶段，如图 6-5 所示。

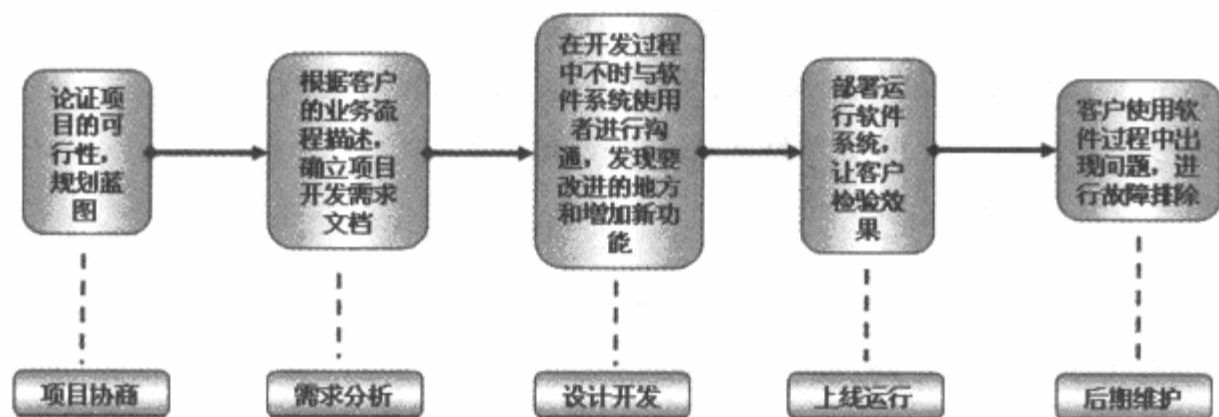


图 6-5 项目生命周期与客户交互内容示意图

在图 6-5 中，除了这五个阶段，在设计开发和上线运行之间，即项目结束后有时还要给客户中直接使用软件系统的人做一些相关培训。在大公司这些可以由专职售后人员做，而在小一些的公司一般由技术开发人员兼职。

同样，在最后一步的后期维护中，排除故障的方式根据公司规模的大小也有所不同。大公司首先由售后部门判断是用户的使用、配置问题还是开发的问题，若是开发的问题，则由开发人员解决。小一些的公司这些工作一般都由开发人员来进行。

### 6.5.2 不要这样对待上帝

身为 IT 人才，面对不懂软件开发技术的客户，多少还是会有些恃才傲物的。还有的开发人员是只懂技术的程序呆子，跟客户打交道的流程漏洞频出，好端端地把到手的客户给气走了。下面就列举几个和客户沟通时的失败案例，作为反面教材引以为戒。

- 不是客户刁蛮，是你没搞清楚就仓促开工。
- 别以为我不懂就忽悠我。
- 别忘了，你不是一个人在战斗。

#### 1. 不是客户刁蛮，是你没搞清楚就仓促开工

“蔡佳娃，随着你的资历慢慢增长，以后你与客户沟通的机会会越来越多，师兄我今天就跟你讲几个不成功的教训，希望你以后不要再犯类似的错误。”

“哇，师兄你太了解我了！”

“首先给你讲个相声里面的段子吧，说有一个包工头接了个工程，规模挺大，还有图纸，上面画着一口水井的图，于是召集自己的人马热火朝天地开始挖井，等工程结束后客户过来视察，结果气得够呛，一分钱都没给就走了。你知道为什么吗？”

“嗯，我知道，是因为那个包工头把图纸给拿倒了，人家是让盖个烟囱，哈哈！”

“回答正确，回到软件开发上来，这就是客户需求和我们设计的矛盾。大多数的客户与软件开发商的纠纷都是从这里引起的。客户认为把自己的需求讲明白了，软件开发方也认为听明白了，结果做出来的却是另一番模样。”

在软件开发行业中，很多时候项目都是这样失败的。原因就是在需求分析设计阶段双方没有

沟通好，对于同一个功能客户和软件开发方的实现概念并不一样。

这种情况应该是软件开发方的失误，开发人员并没有将收集到的客户需求提取出来，翻译成为软件开发的语言，并写成项目文档。所以身为开发人员，在和客户交流的时候必须认真研究其功能需求和生产流程，确保需求分析这个阶段双方思想上的一致性。

## 2. 别以为我不懂就忽悠我

“王经理，这是我们对贵公司项目的说明文档，请您过目。”

“哦，……，我说，我们这个钢铁厂的资源管理系统怎么有两个服务端的设计说明啊？”

“哦，那是因为贵公司需要实现移动端和PC端都可以与服务器进行数据交换，所以需要两个不同的服务端来进行通信，PC端是Tomcat，而手机端需要用Socket服务器，通信协议也是不一样的，当然了，开发成本也会高一些。”

“你们这是想多干点活多要钱吧，蒙谁呢？手机端的应用程序怎么就不可以和Web服务器进行通信啊？你欺负我们不懂是吗？”

“啊？您……怎么……是……”

“在这我也就是个卖钢铁的，可是两年前我也是个干IT的呢！”

故事中的那位IT职场人可算是班门弄斧，泄气到家了。在与客户交往的时候，虽然这种世外高人出现的机会并不多，但是身为技术人员，仗着技术好作假还是比较可耻的。

还有一种需要避免的恶习就是吹牛，反正面对的是外行，把客户吹得晕晕乎乎的正好把项目定下来。可是外行人有外行对不懂事物的执着和坚持，往往信口开河之时许下的诺言会被客户铭记在心，等到项目验收的时候客户啥也不看，就等着技术人员为其演示当时说好的效果。如果无法信守承诺，实现不了，客户心理落差太大，自然会在项目结款方面处处刁难。

## 3. 别忘了，你不是一个人在战斗

“赵经理，这就是我们对贵公司办公自动化项目的可行性论证，请问您还有什么功能需求上的问题需要咨询吗？”

“嗯，我说，我们这个项目你们需要多少人来干啊？”

“这个嘛，大概十三四个吧。”

“那需要多长时间呢？”

“也许是3个月吧。”

“哦。”王经理皱皱眉头，“那这个服务器可以同时支持多少人在线访问呢？”

“5万？或者是10万吧。”

“喂，我说你到底懂不懂啊，问个话回答得这么不肯定，你们公司能不能完成我们的功能需求啊？不行早点说话，别浪费大家的时间。”

不能怪赵经理脾气火爆，与他沟通的这位IT职场人说话也太不自信了，支支吾吾，总是用“也许”、“大概”等不确定的词，这样的表现客户怎能放心把项目交给他们去做呢？

所以说，在和客户沟通的时候，要告诫自己，自己代表的是公司，自己不是一个人在战斗，底气要足一些。如果对方提出的问题的确没有永久固定的答案，如服务器同时支持在线人数的数量在执行不同负载的功能时并不相同，这时可以向客户详细地解释这个问题，这样不仅令对方满

意，同时也会让客户对自己公司的技术水平有一个新的认识。

既然说到了代表公司，那么在和客户打交道的时候就要注意自己的言行举止，不可过于忸怩作态或恃才傲物，也不要和客户乱攀交情，称兄道弟玩假熟等。

### 6.5.3 如何对付不可能完成的任务

以客户为主的产业，总有无法满足客户要求的地方，再出名的饭店也有做不出的菜，而在 IT 这个务实的行业中更是如此了。面对着客户的古怪需求，结合着自身的技术极限，身在职场的你如何对付不可能完成的任务呢？

“蔡佳娃，下面我们谈谈和客户过招的最后一个问题，那就是如何面对自己接不了的项目。”

“啊？接不了在一开始就应该拒绝吧？”

“这就不一定了，一般情况下把客户拉过来和我们谈生意的都是公司里面主管市场的部门，他们有时对技术不了解，为了创造效益，什么都往回拉，所以出现这种情况也就难免了。”

“说得也是啊，那应该怎么办呢？该放弃吗？”

“怎么可以呢，每个客户都是公司的一笔财富，不管是金钱上的还是人脉上的，实在做不来，也不能一点交代都不给的。”

无论是出于什么原因，项目很难按用户的直接要求完成，一般有以下两种情况。

- 根据公司的技术水平和客户的需求说明，项目是可以完成的，但是客户开的价钱不能接受，这种情况下的解决方式要么增加资金，要么减少功能特性。
- 根据公司的技术水平和客户的需求说明，项目实在是无法做出来，但是就像饭店从来不会因为做不出客户要的菜而把客户轰出去一样，这种情况还是需要尽力去争取一下的，步骤如图 6-6 所示。

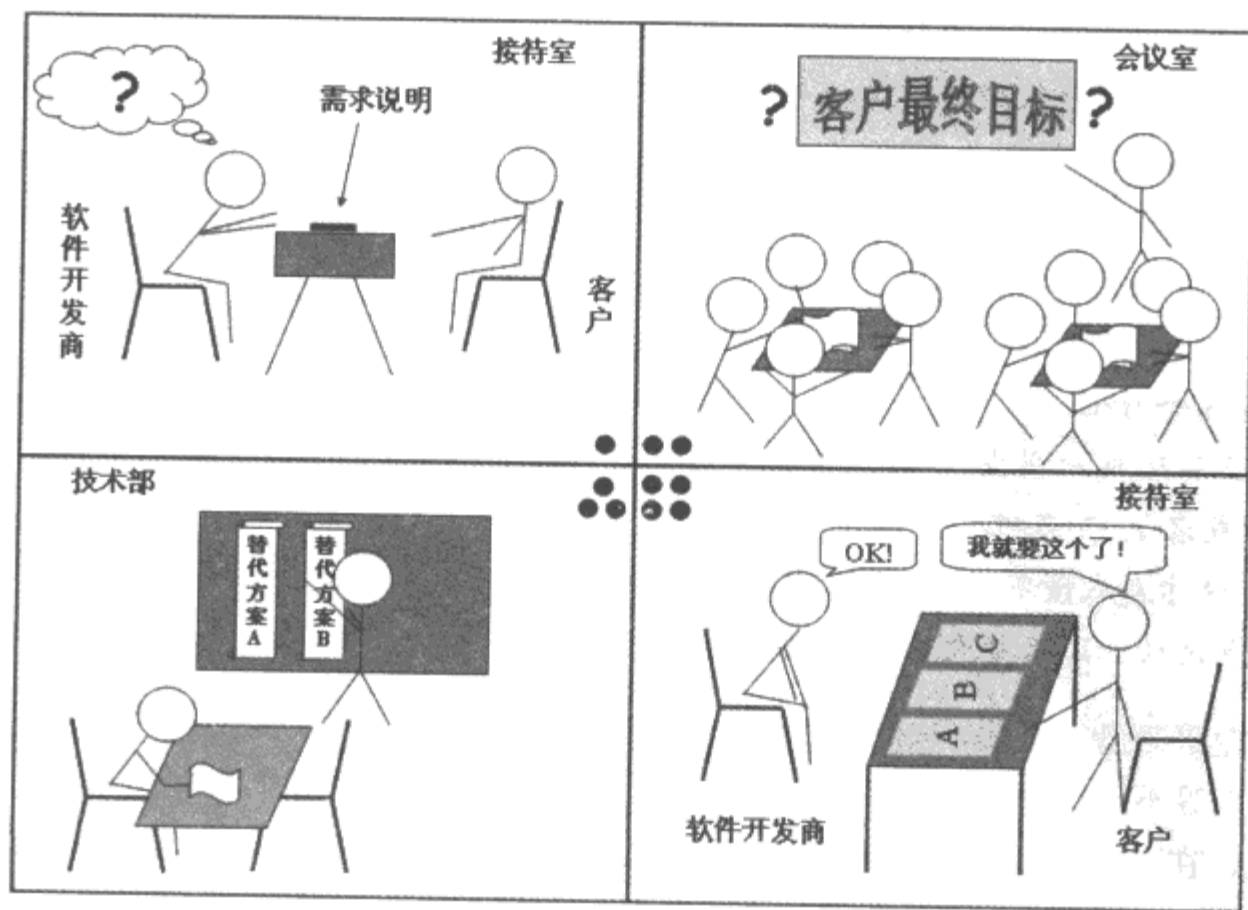


图 6-6 极力争取的步骤

从图 6-6 中可以看出，在接到技术上难以完成的任务后可以采用下述步骤争取。

- 找到客户的终极目标
- 拿出现行技术条件下的替代方案
- 与客户协商

### 1. 找到客户的终极目标

根据客户的需求描述，找到客户最终要达到的目标，往往客户的功能需求都是只停留在表面或是有很浓烈的外行风格。开发人员要学会透过现象看本质，找到用户的真实想法并将其分析转化为软件功能上的本质需求。

### 2. 拿出现行技术条件下的替代方案

根据客户的终极目标，考虑到本公司的最大技术实力，在技术上进行替代性方案的设计。例如发现用户想要一个古怪界面的本质目标是在一个窗口中尽可能多地显示内容，那么就可以采用选项卡控件的方案或是 CardLayout 的布局方案。

### 3. 与客户协商

做出替代性方案后可能有以下三种结果：

- 第一种结果是无法找到可行的替代性方案，进行到这一步，只好跟客户说明，放弃承接项目；
- 第二种结果是找到了几种替代性方案，这些方案虽然在实现上有差别，但是在效果上是殊途同归的，这时就要向其介绍几种方案的优劣，想办法说服客户放弃原来的想法，选择自己的替代方案；
- 最后一种结果就是研究来研究去发现客户的功能可以凭借自己的实力用更好的方案来实现，这种情况下问题就比较好办了。

## 6.6 学着处理和 MM 的关系

谈了这么多职场中做人的问题，好像仅仅把性别限定在了男性，就连本书的两个主人公蔡佳娃和牛开复两个人也都是男同志。但是就像一部电影中所讲的：“这个世界不是男人就是女人”，随着社会的发展和女性在 IT 职场中的不断开拓进取，IT 江湖中的女侠也开始频频出现，暂时还保持着人数优势的男性开发人员就要学着和这些不让须眉的巾帼好好相处了。

### 6.6.1 这个行业的男女比例

“师兄啊，你觉得从事咱们这个行业的 MM 多吗？为什么我们公司里面这么少啊？”

“哦，是吗？从中国的 IT 现状来看，IT 行业中的男女比例差距并不是很大，大约为 3:2，不过很多都不是从事编程开发工作的。”

“哦，那我们公司算是少的了，还不到四分之一，不过有两个是搞编程的呢。”

“一般来说，IT 职场目前还是以男性为主导力量的，女性在职场多半从事技术顾问、销售、行政、人力资源等职务，而且在这些方面 MM 都有着很大的优势，因此很多 IT 公司都乐于在这些领域招聘女性员工。”

IT 职场中女性所占的比例没有男性高，并且从事的方向也不同。原因有很多，主要是因为女性对于 IT 职场中不时加班的工作强度不太感冒，没有哪个 MM 希望总顶着个熊猫眼。

同时在 IT 行业中，由于没有很多知名度非常高的可以比肩比尔·盖茨的女英雄作为榜样，因此青睐这个行业的 MM 也就少很多了。其次，思维模式和做事方式的不同也使得 IT 职场中男性和女性从事的方向有所不同。

不过就像本书之前讲过的，女程序员是最先出现的。随着广大 IT 公司招聘规则的改变和当代教育的发展，女性也在重新回到曾经属于自己的土地上准备和男性展开争夺了。

### 6.6.2 如何面对异性员工

对于女性 IT 员工来说，既然职场上占绝对优势的是男性，那么女性员工向男性员工学习的地方就应该不少，比如对待工作的态度、敢于接受挑战的胆量、适应能力强的特性以及不拘小节的性格等。这样会大大增加女性在职场的竞争力。而对于男性 IT 员工而言，女性员工所特有的做事小心不急躁、定力好、善于沟通等品质也是需要男性员工去学习和改进的。

“师兄，那我们之前讨论的职场的做人道理到了和女员工相处的时候还适不适合啊？”

“基本上还是适合的，就是需要再额外补充些。”

“哦，那需要补充些什么啊？”

“那就是在 IT 职场上，男性员工和女性员工擅长的方面不同，男女双方都应该利用自己性别上的优势去帮助别人。比如男性员工更能吃苦，能熬夜，做事也更有热情；而女性做事细心有耐力，考虑问题也周到，所以男性和女性员工应该在工作上互相补充，使自己的团队更具有战斗力。”

在漫漫职途中，不可能只和同性打交道，几乎没有一个行业是只有一种性别的员工存在的，IT 行业也是如此。所以在职场中学习为人处世经验的同时，也要不断积累与职场异性的相处之道，让自己在做人方面成为一个全才。

## 6.7 本章小节

前面一章讲述了在职场中做事的正确方法，本章讲述了在职场中为人处世的哲学。为人处世的智慧不是看看书就可以学会的，需要每个人在与他人交往的时候仔细体会，寻找最适合自己的处世之道，为自己在职场的奋斗保驾护航。

做人和做事的学问是每个开发人员都必须掌握的，是同等重要的个人财富。IT 职场犹如江湖，闯荡其中既要勤于修为，练就一身绝技，又要广泛结交武林豪杰，尽知人情世故，这样才可以让自己的江湖之路更加顺畅，成长更为迅速。



# 第 7 章 百尺竿头，更进一步

“百尺竿头不动人，虽然得入未为真。百尺竿头须进步，十方世界是全身。”这虽然是佛家的偈语，但阐述的道理却具有很大的通用性。在 IT 职场打拼，顺利度过了菜鸟阶段，只能勉强算是到了“百尺竿头”的地步，虽然已经入门且略有所成，但是仍然没有真正达到高手的境界。处在这个位置不能自满，也不能松懈，只有不断地“更进一步”，才能见识到 IT 行业中的最高境界。

## 7.1 技术不是万能的

对于很多开发人员来说，努力工作提高技术水平是第一位的，但是光靠技术这一条腿走路是不行的。要想成为一名业界高人，必须认识到 IT 行业的最终目标是服务于客户，而且为了实现这个目标还需要补充 IT 技术之外的知识。

### 7.1.1 为何 IT 是个服务业

前面的章节本书也曾提到过 IT 行业本身是服务业，对于这一点很多读者可能会感到疑惑：为何如此叱咤风云、技术至上的 IT 行业却是要服务于他人、看他人脸色的呢？

“师兄，你说我们 IT 行业算是第几产业啊？”

“那你说应该是第几产业呢？”

“我觉得应该是第二产业吧，跟汽车制造、钢铁工业差不多吧？”

“错啦！IT 属于第三产业！”

“那不就成了服务业了吗？怎么会呢？”

原因就是以技术为主的 IT 行业一般很难直接实现自己的价值，必须将技术结合实际需求做成产品，运用到各行各业中去才算真正发挥了作用。而那些使用了 IT 技术产品的企业，就会享受到 IT 技术所带来的诸如生产效率的提高、企业效益的增加之类的好处。

比如有了文字处理软件，就可以做出绚丽多彩的排版效果；有了财务报表软件，就可以把广大会计师从烦琐浩瀚的账目中解救出来；有了生产监控系统，就免去了生产线上的员工在危险环境下操控和监测设备……正是这些应用服务的提供，才真正实现了 IT 技术的价值。因此 IT 产业是个不折不扣的第三产业，以最神奇精彩的魔力改变着几乎所有其他的产业。

当然了，IT 行业也会为 IT 本身服务。比如一些应用服务器，像 Tomcat、WebLogic 之类的，将会在具体的面向终端客户项目的开发中被采用，以便更快捷有效地进行终端客户项目的开发。

### 7.1.2 业务流程要清楚

如果没有技术水平是万万不能的，但是仅有非常高的技术能力也不是万能的。既然 IT 是服务业，那么就应该是客户需求驱动的。想要让客户满意，就必须对客户所在的行业领域有所了解。

否则闭门造车，或者虚掩着门造车，都无法使自己的发明创造驶入正轨。

“师兄，半年前我们公司给一个炼钢厂做项目，最后失败了，现在公司正在想办法补救，把项目尽量完成。”

“那是为什么失败啊？”

“大家埋头开发了半年，结果做出来以后客户却不能用！”

“哦，看来你们公司在项目开发前对这个钢铁厂的生产流程没有研究透，结果开发出来的项目无法满足企业的实际应用需求。”

“不过，我们搞软件开发的，有必要研究那个吗？”

搞软件开发的，当然有必要研究客户行业的业务流程。很多时候决定一个项目成败的，不是软件开发方的技术能力，而是对于项目目标客户所在行业领域知识的了解程度。很多软件开发商和客户之间的纠纷就是由此而来的。

每个行业的生产或业务流程都是经过很长时间的探索和总结而诞生的，软件系统一般只是在此基础上进行不同程度的优化和改进。如果软件开发只研究技术，凭自己的臆想进行产品开发，那么做出来的成果必然无法成功交付使用。

“师兄你说得也很有道理啊，我们公司为了那个烂尾项目，现在命令开发团队狠命研究冶金炼铁的工艺流程，好像还请了附近大学的一个教授呢。那师兄你给我详细讲讲对于开发人员来说这些技术之外的行业知识都是什么吧？”

“好，首先可以将这些技术之外的东西分为业务流程、专业知识、系统操作方式设计等几个方面，下面我们首先来谈谈业务流程知识。”

很多所要开发的项目目标是实现业务流程的管理，如学校学生信息管理系统、火车站售票系统等。业务流程是指企事业单位进行正常生产活动所涉及的一系列相关联的活动的整体，业务流程讲究信息数据流通的方向和次序，类似结构化的程序，输入数据经过计算输出结果。

“首先要谈的是业务流程知识，你们公司的那个钢铁项目就属于业务流程这一类的。”

“那什么是业务流程啊？”

“最简单来讲，业务流程就是做事的方法和顺序，比如你去火车站买票，你先告诉售票员你要买的车次和票数，然后售票员收你的钱，把票给你，这就是一个流程。”

“那也没什么难的嘛。”

“我刚说的是理想情况，如果某个车次只剩一张票而多个系统终端都要买该如何分配？如果顾客只知道起始车站怎么办？如果有乘客退票呢？或者学生买票？”

“哦，看来需要考虑的事情还真多啊。”

“我给你讲讲我两年前待过的一个公司，那时我在那个公司算是个技术总监，当时接了一个医院的管理系统。我让几个开发人员着手去做，结果进行到一半给客户验收的时候，客户说用这套系统根本不符合医院的正常工作流程。”

“是不是他们没有深入研究医院的业务流程啊？”

“那几个人认为自己看过病，就明白了医院的流程，根本没去做研究，结果弄成这样。后来我只好去医院待了一个月，彻底把这套流程弄清楚了，回去开发出的产品就非常符合实际情况了。”

开发与业务流程密切相关的系统时都要仔细研究目标的业务流程，这样才能保证开发出来的产品能够符合实际情况并直接投入使用。很多时候最难的就是搞清楚业务流程，看懂业务流程后，技术实现就根本不是问题了。

### 7.1.3 专业领域的知识要了解

在实际开发中还有一类项目其目标是给特定领域的专业人士一定的辅助，提高生产效率，如财务账目、建筑预算、计算机辅助设计等。这一类项目与业务流程密切相关的系统不同，基本上每个客户的需求都是基本相同的。这类项目的成败，很大程度上取决于对目标专业领域知识的了解程度。

“师兄，业务流程方面的问题我大概明白了，那另外一个要谈的方面是什么呢？”

“第二个方面就是专业领域知识，这个跟业务流程可不一样，业务流程需要你仔细观察，看会了再模拟实现，而专业领域知识就需要你像学习新技术一样去钻研了。”

“那都是什么样的项目需要专业知识啊？”

“最常见的一类就是财务系统了，我就曾经开发过这类项目。说到这我想起来当年的那个同事，开发财务软件的时候居然把增删改查四项功能全部添加到账目管理系统中。”

“嗯？不对吗？数据的基本操作不就是增删改查吗？”

“哎呀，看来我的小师弟也会犯这样的错呢，不过不怪你，你对于会计学不了解。一个单位的账本怎么能够删除和修改呢？你从财务处取了1000块钱，回头改成100或者直接删掉吗？按照你的思路进行开发，软件还没交付使用呢，警察叔叔先过来看望你了，这都是违法的啊。”

“是吗？那如果出现记账错误怎么办？”

“会计学中用蓝字冲销和红字冲销两种方式对付错账，简单来说，如果一笔账记错了，则先用一个红字冲销冲掉错误账，再用一个蓝字冲销记录正确的账目即可。肯定是不存在把账目调出来修改或删除的，所以开发财务软件的时候如果不了解相关领域的知识可是要闹笑话的。”

其实对于开发人员来说，有些时候所开发项目的专业领域知识才是整个项目进度的瓶颈，而不是技术。因此，开发具有专业知识的软件时需要对该领域的知识有一定的了解，不一定特别深，但至少够用才行。如开发一个建筑预算软件，想要保证软件的可行性与实用性，将几百页的国标建筑规范通读一遍，一点也不过分。

从另一个角度来看，开发的软件项目多了，自己接触的各行各业的知识也就会越来越多，逐渐会把自己打成一个“全才”。因此，这种技术之外的积累也是一笔可观的财富。

### 7.1.4 软件系统的操作方式

软件系统的操作方式，或者叫人机接口，也是在项目开发中需要考虑的一个重要方面。因为一个软件系统最终是要由人来使用的，其操作方式决定了其使用的效率。所以在开发中不仅要注意技术水平上的创新，人机接口的设计也是不容忽视的。

“蔡佳娃，你玩过游戏吧？”

“玩过啊，不过我玩游戏和现在的工作一样，都是个菜鸟，呵呵。”

“那你应该知道一些高端玩家都是怎么操作游戏的吧。”

“看过视频，那些高手 PK 都是用的专属键盘，玩得那叫一个快啊！一点也不亚于弹钢琴的。怪不得好多流行的游戏都被称为‘键盘杀手’呢。不过，师兄，这个跟我们要讨论的有关系吗？”

“有关系啊，从玩游戏你大概也能看出来，要想快速操作一款软件，还是得靠键盘。”

很多人都去火车站买过票，排队的过程是很漫长的，但这并不是售票系统的原因。仔细观察就会发现，火车站的售票系统大都是不用鼠标的，或者干脆是没有鼠标的。售票员的操作完全靠键盘，这样的效率已经非常高了。想想看，如果把火车站的售票系统改为鼠标操作，每买一张票都要来来去去地点，那售票窗口外面排的队不知道要有多长呢。

同理，很多像银行、超市等地方都对工作效率要求较高，在设计这些专用系统的时候，就应该只用键盘进行操作。或者至少多用键盘操作，这样才能够提高服务速度。否则，一个不符合实际应用的人机接口，是无法胜任高效率工作的。

“师兄，既然是这样，那我们在平时使用各种软件的时候怎么没感觉到用键盘操作的需要啊？”

“平时又没有人要求你多少秒售多少张票或开多少发票，你当然不会感觉到。而且键盘操作也不是天生就快，需要经过训练才能掌握键盘上每个按键和组合键的功能。而鼠标你再怎么训练，也不可能达到这种速度的。”

“是啊，快速操作也是有代价的。”

“像我们平时做软件开发使用的开发工具软件，一般也不会是纯键盘操作。因为不像售票系统的固定流程，编写代码是需要脑子不断思考的，所以整一堆快捷键对于开发效率不会有太大的提高，倒是复杂缺乏人性化的操作方式可能会招来怨声一片。”

不同的用户群对于软件系统的操作方式有着不同的需求，因此在设计软件系统的时候，要根据实际情况进行恰当的人机接口设计，适合的才是最好的。

## 7.2 书是人类进步的阶梯

在学生时代，书一直伴随着每个人的学习和成长。步入 IT 职场，随着一个开发人员资历的增加和职位的升迁，很多人都慢慢忽略了读书的价值。这对于个人的发展是很不利的，越是搞技术的，越是要学习，而读书则是学习中最传统和最有效的方式。而面对着如此繁荣的图书市场，选一本看了确实能获益的好书，对于所有人，尤其是初学者来说，都是一个值得研究的问题。

### 7.2.1 还要不要读书学习

“读万卷书，行万里路”，这是很多人在上学的时候学到的一句俗语。这句俗语说的是清初思想家顾炎武先生以骡马载书自随，游学于北方的故事，比喻要多实践，多学习。不过最近网上有了一种新的版本：“读万卷书不如行万里路”，意思是读书无用，实干才是最好的方法。姑且不用管这句话是对是错，这最好不是一名软件开发人员的心态。

“师兄啊，前几天你跟我说身为菜鸟要不断学习以提高自己。昨天我路过书店，想起这事了，进去后发现想挑一本买了不后悔并且对得起定价和自己的书还真是不容易，结果我就没买。不过

我想知道的是读书对于每天需要实战代码的开发人员来说真的有必要吗？”

“那你是不是觉得自己现在用不着学习呢？”

“倒不是觉得自己层次很高不用学习，只是就着自己已经掌握的这些技能，也差不多够每天用了。就算有时间读了书，估计也没机会用啊！”

“看来你的眼光还不是很长远啊！很多像你这样进行固定类项目开发的人员，随着从业时间的增长，技术和经验都提升到一个较高的层次。很多人就会慢慢懈怠，习惯于重复式的工作，脑子也会慢慢地僵化。”

“顺境不懈怠，逆境不沉沦”，这是原全国人大常委会副委员长成思危在应对金融危机时说的一句话，这句话送给每个开发人员都非常合适。不少开发人员的一个通病就是技术提高到一定程度，对于新技术就不再有特别的需要，这种对待“顺境”的方式对自己是非常不利的。

技术积累才是每个开发人员应该不断去做的事，金融危机下的春节为什么中国的饭馆还是有人光顾，而外国的餐饮业已经不能用惨淡来形容？深奥的经济理论不谈，中国人储蓄的习惯算是一个比较重要的原因。对应到 IT，这就是技术积累。


“那师兄，照你这么说，俺们这些菜鸟在努力适应新工作环境的同时，还得注意让自己多补充补充新营养喽？”

“不仅是你们这些菜鸟，每个人都要不断接触新的技术知识。谁也不知道明天太阳从哪里升起，说不准哪一天你优哉游哉地上班去，却发现你的竞争对手推出了具有新理念的产品。到那个时候才想到采用新技术去赶超对手，恐怕就太晚了。”

“是啊，只怕到时候黄花菜都凉了。”

“勿临渴而掘井，宜未雨而绸缪”，行万里路的实干精神固然可嘉，但是放眼于未来，锲而不舍地破万卷书会让自己对于未来更加成竹在胸。微软总裁比尔·盖茨家中就藏书万卷，而且大都阅读过。

很多 IT 巨头都是靠着对新技术的准确把握而走向成功的，而现在，市面上刚刚开始崭露头角的 Lucene 和 Nutch 搜索引擎技术、Hadoop 分布式存储、JavaFX 等技术，都是需要开发人员适当了解的。第一个吃螃蟹的人收获最多，谁能说这些新秀技术没有可能成为下一个 IT 骄子呢？

 **提示** 本节中提到的一些新技术各位读者可能都不太了解，不用担心，本书后面的章节将会对这些明天的骄子有所介绍。

### 7.2.2 选本好书不容易

读一本好书很让人受用，但是选一本好书的过程就不那么容易了。在漫漫技术书籍的浩瀚海洋中，究竟哪一本是对自己最有好处的呢？

“师兄，我明白了，看来我以后也要挤出时间继续读书了。”

“嗯，下面接着谈如何选本好书的问题，这个问题和为什么读书一样重要。”

“是啊，我上次路过书店没买到书就是因为 Java 技术的书实在是太多了，我只是随便转了转，根本没法挑，也不知道该怎么挑。”



“就现在的情况来看，选书的学问有时候比看书都大。因为市面上的好书和坏书都有，经典书和普通书相当，国内版与欧美版共存，教学用书和实践手册齐上架，确实是个繁荣的市场啊。”

“那师兄你说该如何选本好书呢？”

“首先指出很多人买书的一个倾向，那就是买国内作者编写的技术书籍，这倒不是不对，只是他们这么做的动机却不很正当。这些人买国内作者的书第一是因为它们是中文写的，第二这些书一般都比较薄，看起来更有信心一些。”

在选择一本书的时候，一定要记住“好书无薄书”这个原则，这里的薄不是单纯指页数上的薄，而是综合书中所包含的内容和页数来评价的。比如一本区区 400 页的 Java 技术书从 Java 诞生背景一直讲到 EJB 的开发，这就是真薄书，而一本花 300 页的介绍集合框架技术的书就不能算是薄书了。

再谈谈“厚”的问题，一般来说综合性的技术书籍都是比较厚的，知识层次也是比较深的。而一些薄的综合性书籍虽然里面的知识一学就会，但是含金量太低，大都是些“Hello World”之流，严重与实际开发脱轨，学会了也没有多少用武之地。

“师兄，那国内和国外的书有什么差距吗？”

“国内的技术书籍，有一部分是由于厚薄和知识层次的问题，在内容上讲的并不是很透彻。有的时候甚至讲的是错的！”

“啊？还会讲错？”


“是啊，我记得 2007 年帮别人买了一本 Java 语言程序设计的书，就顺手翻了翻，在看到‘继承与覆盖’这一节的时候，居然发现里面印着这样一句话：子类无条件地继承父类的不含参数的构造函数！白纸黑字，吓了我一跳。”

“是啊，的确是错的啊，继承怎么能用来描述构造器呢。”

“所以说如果买了这些书，没学会还好些，如果学会了才真是‘挥刀自宫’啊！”

“师兄，看来想买本好书还真得小心挑选。”

“不过你要记住了，买书是为了看书，看书是为了掌握书中的知识。既然你下决心买一本书，那么就别想图省事，必须做好吃苦的准备才行。”

 **提示** 其实，随着这几年中国 IT 行业的迅猛发展，中国的 IT 技术书籍在质量上提高了很多，而外国也开始有一小部分的书籍涉嫌假、大、空的内容。所以在选书的时候不能过分依赖国内或是国外。国内一些内容比较好的书有《Java SE 6.0 编程指南》、《深入浅出 Hibernate》，国外的一些经典书有《Thinking in Java》、《设计模式：可复用面向对象软件的基础》等。

## 7.3 解决问题的方法

身为开发人员的主要工作，其实就是在不断地用自己的代码解决一个又一个的问题。解决问题的方法很多，本书将这些方法分为正招和歪招两种。熟练地运用这两种解决问题的方法，对于一个开发人员的成长是很有裨益的。

### 7.3.1 正招和歪招

“蔡佳娃，今天我来教你些实用的东西。教你一套解决问题的方法——正招和歪招。”

“哦，那是什么啊？正统的招数和旁门左道吗？”

“不对，正招和歪招不是这个意思。它们体现的是一种灵活解决问题的能力，这种能力是每个软件公司都喜欢的，也是客户所青睐的。”

“是吗？那师兄你赶紧给我讲讲吧。”

#### 1. 正招

“正招指的是在解决一个问题的时候，市面上已经存在成形的解决方案、平台或算法了，这时候可以直接拿来为我所用，这就是正招。”

“哦，正招就是采用正统成熟的招数呗。”

“对啊，正招大家都用着没问题，所以我们也来用。反正方法都是共享的。”

“这么说来，正招用起来多舒服啊，直接吸收前人的优秀成果。”

“不要这么得意，很多时候你都是有眼不识泰山的，想要用好正招也不是那么容易的。”

正招是开发人员在工作中经常会使用到的，比如在开发过程中遇到需要排序的问题，那么就没必要自己闷头去研究一种排序算法了，市面上存在的排序算法可谓比比皆是，而且都比较成熟可靠。这时挑选一种符合开发要求的排序算法就显得高效很多了。

虽然正招用的时候是手到擒来，药到病除，但是要想完美地使用正招必须要对所在行业的知识了解得相当透彻和全面，也就是说很多时候开发中不是缺少正招，而是缺少发现正招的眼睛。

#### 2. 歪招

“师兄，那歪招是什么呢？”

“正招虽然好用，但是很多时候面对的问题并不是简简单单用正招就能解决的，打个比方说，没有客户会跑来让你写个快速排序法的程序然后验收通过，啪，甩给你十万块钱的。”

“是啊，客户的要求可是很让人眼花缭乱、应接不暇啊！”

“所以说，在面对某个问题一筹莫展的时候，就该歪招出马了。首先要说明的是使用歪招不代表这个人的人品不好，歪招也不是邪门歪道。歪招是指市面上并没有针对某个问题的成熟统一的解决方案时，利用所学的知识和技术，将它们进行改动、调整和拼凑，以满足客户需要的一种方法。”

“看来歪招只是名字不好听而已，还是非常实用的呀。”

能适当使用歪招是一名软件开发工程师最重要的一种能力了，歪招不是凭空臆想，也不是单纯的几种技术的拼凑和搭配，而是需要进行再次加工和创造的。歪招算是一种创新，而且如果歪招经过很多人的千锤百炼，也会成为正招来供人使用。

正招是歪招的基础，要想驾驭好歪招必须对正招有足够的了解和把握。不论是正招还是歪招，都需要对所学的知识深刻掌握，对所在行业高度了解才行。

### 7.3.2 优先使用正招

既然解决问题的方法有正招和歪招之分，那么作为一名开发人员，在进行项目开发时的第一个原则就是优先使用正招，这是解决问题最有效的方法。

“师兄，我们在开发过程中应该是先往正招这方面考虑吧？”

“嗯，我在参加很多 IT 界的会议讲座时，经常听到的一句话就是：不要重复发明车轮，意思就是某件事情的解决方案已经很成熟和完美了，所以遇到此类问题的时候应该直接享受正招带来的便捷，在已经存在车轮的情况下进行重复研究是没有价值的。”

“是啊，重复发明车轮也不一定能研究出来呢。”

“为了加深你对正招的理解，我来举几个案例，让你明白正招的威力。”

#### ● 案例一

“这个项目案例的描述是这样的：需要对 TB、PB 级别的数据进行分布式存储。别看这个项目的描述很简单，解决起来可是非常困难。”

“哦，那是为什么呢？”

“对于知识面不够广的开发人员来说，这时的解决方案有两个：一个是买，如 Sun 公司就提供这种服务，不过昂贵的价格不是一般公司可以接受的；第二个方案就是自己做，那需要面对的难题可就更多了，需要考虑数据的同步和冗余问题，还需要保证稳定性等。”

“哦，那看起来这个项目无解了？”

“非也，解决这个问题还是有正招的，那就是 Hadoop 技术，这个开源技术完全可以解决上述问题。不过如果不了解这个正招，可就真的是一筹莫展了。”

“看来真的是正招出马，一个顶俩呀。不过能想起来去用它，才是最难做到的啊。”

Hadoop 是 Apache 基金会研发的一个开源软件平台，是由 Java 开发的。Hadoop 提供了稳定可靠的接口，开发者可以基于 Hadoop 开发分布式的应用程序，从而可以对海量数据进行存储和处理，同时也能很好地保证数据的可靠性和稳定性。因此，在解决这类复杂问题的时候，Hadoop 技术就显得非常有优势了。

一般来说，在面对一些比较大的问题时，普通的开发人员只能去寻找正招，这样能保证研究出来的解决方案的可行性与可靠性。

#### ● 案例二

“蔡佳娃，你经常上网，应该对于网上的酷炫界面见得不少吧？那你知道这些是如何开发出来的吗？”

“是啊，现在的 Web 界面是越来越漂亮了。这些好多都是 AJAX 技术的 Web 应用吧？”

“嗯，差不多，那我们以一个简单的视觉效果为例，鼠标悬停于某个图标之上的时候，图标会放大，这个效果的开发方式你觉得是怎样的？”

“AJAX 不就是主要采用 JavaScript 嘛，直接用 JavaScript 开发不就行了？”

“照你这种想法，那么 AJAX 的开发社区中可真是各式各样的轮子满地跑了。你知道吗？如果这个效果从头用 JavaScript 开发，没有一千多行代码是办不到的。何况开发出来还可能会有 bug。”

“既然是这样，那有什么正招吗？”

“正招当然有，针对这个例子，采用 AJAX 的一个开源工具包 Dojo 就可以做到，不到十分钟的工夫就可以出效果。这些正招都是那些高人们研究和总结出来的。”

DojoToolkit 提供了很多的 JavaScript 部件，同时也支持很多 Widget（一种与用户界面有关的组件平台）的实现。因此开发人员使用 Dojo 来开发的时候就会省去很多工作，使得开发过程更加顺畅。比如案例中的鼠标悬停，图标放大的效果就是 Dojo 众多 Widget 中的 FishEye 效果。这个效果用 Dojo 开发是相当快捷的，用 Dojo 开发的 FishEye 示例如图 7-1 所示。



图 7-1 Dojo 的 FishEye 示例

### ● 案例三

“下面我们再举一个使用正招的例子，蔡佳娃，你应该了解传统 Web 应用的通信规则吧？”

“知道的。客户端发送请求，Web 服务器收到后经过分析处理再把数据传回客户端，基本上是遵循你来我往的原则。”

“嗯，差不多是这样，在这种通信规则中，客户端有了变动可以通知服务端，那如果服务端变了呢？又该如何告知客户端呢？”

“那就定时刷新呗。这算是个歪招吧，嘿嘿。”

“哎，你这个方法连歪招都算不上，定时刷新的话如何确定时间间隔呢？太快了屏幕总是在闪，而且并不是每次刷新都有价值，还增加了系统的负载。而如果刷新的频率太慢会丢失很多中间数据，比如一个生产环境的数据监测图如果刷新过慢则会产生严重失真（如图 7-2 所示）。 ”

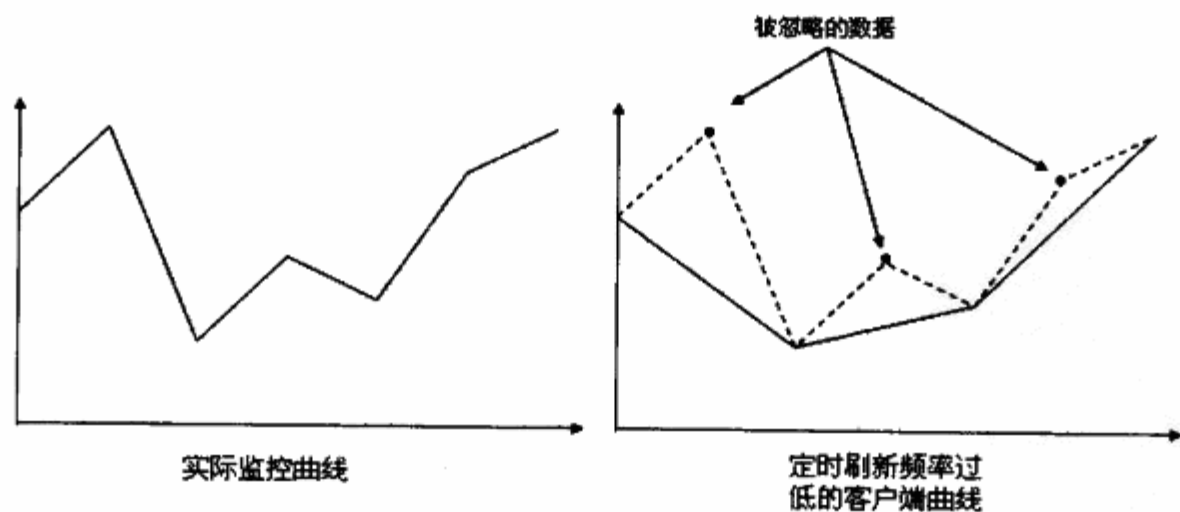


图 7-2 定时刷新频率过低的客户端

“哦，说得也是。那有什么正招来解决这个问题吗？”

“正招就是服务器推送技术 Comet，采用这个技术，服务器端如果发生变化，则会立刻告知客户端，这样就真正实现了数据的实时交互。”

Comet 技术是比较常用的服务器推送技术，原先对实时性要求高的一些应用都是采用定时刷新，而 Comet 技术则很好地满足了许多实时性应用的需要。作为一个开发人员尤其是 Web 开发人员来说，Comet 技术的发展大大丰富了 Web 应用的开发内容，了解并掌握 Comet 这个不可多得的正招非常有必要，Comet 技术的示例如图 7-3 所示。

股票编号	股票市值	涨跌%	曲线颜色	是否绘制
10000	24.02	+8.46 % ▲		<input checked="" type="checkbox"/>
10001	46.45	-8.92 % ▼		<input checked="" type="checkbox"/>
10002	15.38	+6.08 % ▲		<input checked="" type="checkbox"/>
10003	46.63	-1.95 % ▼		<input checked="" type="checkbox"/>
10004	41.42	-7.0 % ▼		<input checked="" type="checkbox"/>
10005	52.24	-8.73 % ▼		<input checked="" type="checkbox"/>

图 7-3 Comet 技术示例

### 7.3.3 正招不够，歪招也可以上

正招使用起来的确是立竿见影，但是就像医学上会出现翻遍医经药典也找不到治疗方法的疑难杂症一样，开发人员在解决问题的时候，也会遇到找不到成熟完美的正招的情况，或者是问题的规模过大，已经超过了正招的能力极限。这个时候，谁能够首先拼凑搭配出一个歪招将问题解决，谁就在成功的路上进行了一次飞跃。

“蔡佳娃，现在我们来谈谈歪招。歪招是在正招这条路走不通的情况下采取的对策。如果问题用歪招成功解决，那么很有可能未来这个歪招就会变成正招，因为它在解决这个问题上有权威。”

“是啊，那歪招是怎么诞生的呢？”

“我以我曾经做过的一个项目为例，来加深你对歪招的理解吧。”

“好啊，好啊，让我跟着师兄学几招歪招。”

“呵呵，具体的项目背景我就不提了，直接来看抽象后的数学模型吧。”

在一次运算当中，需要对一个含有 15 个参数的方程进行探测，15 个参数取 0 到 7 之间的随机数，每次探测一旦得到满足条件的值，则运算结束，输出 15 个参数的值。在所有组合情况中，满足条件的随机数组合大概占到 1% 左右。

#### 1. 寻求正招

“如何？蔡佳娃，你的想法是怎样的啊？”

“这个嘛，可以这样吧，既然是找到一个满足的解就可以，那就写个穷举的 for 循环呗，15 个参数就写 15 层 for 循环，这算是个正招吧。”（代码如下所示）

```
1  for(i1=0;i1<8;i1++) {
2      for(i2=0;i2<8;i2++) {
3          for(i3=0;i3<8;i3++) {
4              .....
5                  for(i14=0;i14<8;i14++){
6                      for(i15=0;i15<8;i15++) {
7                          //业务逻辑代码 .....
8                      }
9                  }
```




```
10          .....
11          }
12      }
13 }
```

“我一开始也是这么想的，这的确算是解决一般问题的正招。不过写完运行之后，半天没出来结果。我说的半天指的真是半天，从早上一直到晚上，机器仍然没有运行出结果。”

各位读者可以自己用 C 语言或 Java 写一个嵌套 15 层 for 循环的例子，循环最里面不做复杂的运算，只做如 “a=1;” 这样的简单操作，看一看自己有没有耐心等到程序运行结束。

了解汇编知识的读者都应该知道，计算机在执行循环语句的时候，是不断地在内存中进行跳转操作的，所以深层次的循环嵌套必然会将大量的 CPU 时间浪费在跳转中。真正在实际开发中，循环嵌套一般是不会超过三层的。下面的两个代码片段，虽然执行 “a=1;” 操作的次数相同，但是执行速度却相差数倍。

```
1  //代码片段 1
2  for(i1=0;i1<10000;i++){
3      a=1;
4  }
5  //代码片段 2
6  for(i1=0;i1<10;i1++){
7      for(i2=0;i2<10;i2++){
8          .....
9              for(i10=0;i10<10;i10++){
10                 a=1;
11             }
12         }
13     }
14 }
```

 **提示** 读者若准备做上述实验检查运行效率差异最好选用 C 语言，因为 Java 编译器的自动优化非常厉害，上述代码中只执行 “a=1;” 的简单操作很可能经编译器优化后执行效率差不多。不过从这点也可以看出，Java 已经是十分成熟的开发平台了，非常有助于水平不太高的人开发出优质代码。

## 2. 拼凑正招

“那师兄，这个运算速度这么慢，肯定不行吧？”

“这个不是慢的问题，是有没有结果的问题。如果超出一定时间还没有运算结果，客户就已经把它定义为行不通了。”

“是啊，那你是怎么解决的呢？”

“后来我就想了，既然解的分布是没有规律的，那我就不按照顺序找。在循环的过程中不按照顺序来，而是在每一层循环中随机选取一个数，只要它没有被选中过，就继续到下一层继续选数，选够 15 个之后就进行探测。”（代码如下所示）

```
1  for(i1=0;i1<8;i1++){
```

```

2      //产生随机数，并检查随机数是否重复
3      for(i2=0;i2<8;i2++){
4          //产生随机数，并检查随机数是否重复
5          .....
6          for(i14=0;i14<8;i14++){
7              //产生随机数，并检查随机数是否重复
8              for(i15=0;i15<8;i15++){
9                  //产生随机数，并检查随机数是否重复
10                 //业务逻辑代码，进行探测
11             }
12         }
13     }
14 }
15 }

```

其实要分析出上述结果并不难，联想到现实生活中，如果有一筐苹果，其中有1%的苹果是坏的，要求从筐里找到一个坏的即可。有的人可能会按顺序一个一个把苹果拿出来看是不是坏的，但是估计有更多的人则抱着碰运气的态度到筐里随便拿出一个检查，很多时候都是后者效率要高一些。

再仔细研究这段代码的情况，15个参数，每个参数可取8个值，那么一共就有 $8^{15}$ 种情况，而根据题目说明， $8^{15}$ 种组合中仅有1%满足条件。这种在很大的样本空间中寻求最优解的问题显然用穷举法效率太差，必须想办法改进。改进方案就像从筐中找烂苹果一样，随机寻找。

其实很多著名算法都是从日常生活和自然规律中提炼出来的，比如遗传算法就借鉴了遗传学的优胜劣汰，而蚁群算法则是通过模拟蚂蚁寻路的行为模式而提出的。

### 3. 产生歪招

“师兄，那这种改进算法解决问题了吗？”

“还是不行，虽然比上一个办法好，有时等一段时间会出结果，但是客户要的不是有时，所以这样还是不行。后来我想到了，根据我的算法，我想要做的就是产生15个随机数，然后进行探测。当时我刚刚看完手头一本汇编语言的书，立刻我就有了想法。”

“什么想法啊？”

“我这次只用了一层循环，每次产生一个64bit的随机数，验证没有重复后，用位运算将其每3个bit位切出来作为待测参数（见图7-4），这样一次就产生了15个待测参数，再进行探测，这样的效率就会大大提升了。”（代码如下所示）

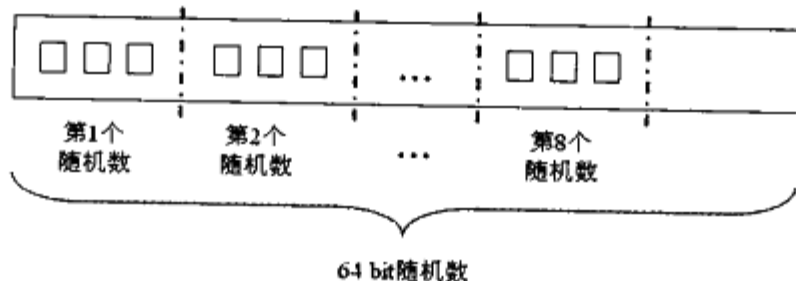


图 7-4 随机数切割图

```

1  while(true){
2      //产生 64bit 随机数，将其每 3 位切割成一个 0 到 7 的数，取 15 个

```

```
3      //遍历探测过数组，查看此随机数组合是否出现过，出现过则进行下次循环，否则继续
4      //业务逻辑代码，探测
5      //将产生的随机数添加到已探测过的数组中
6  }
```

“那师兄，这次应该成功了吧？”

“我用这种方法写出来以后，试了几百次，总是一点运行马上出结果。于是宣告问题解决！”

“看来，歪招对于解决问题还真是好使呢。”

本小节通过一个真实项目问题的解决过程，向读者展示了歪招在项目开发中所起到的作用。作为一名开发人员，要学会优先使用正招，而歪招的巧妙之处就在于那些微小而又实用的创新。不过，需要明确的是，歪招是建立在一个软件工程师对正招灵活运用的基础之上的。

## 7.4 软件产品的目标

开发一个软件产品的目标有很多，如为了打败竞争对手，取得自己在某个领域的领导地位；或者为了满足客户要求，使服务更周到；又或者是开创一种新的商业模式等。通俗点说，最终目标就是拿到 money，而在通往那个目标的路上，软件产品的次级目标也可以统一成为两个方面：功能与性能。

### 7.4.1 实现功能是底线

软件产品的功能是指软件产品能干什么，而性能指的是干得快不快、好不好。在很多情况下，软件产品的功能和性能是辩证的关系。就像工作和玩耍一样，这边有“业精于勤荒于嬉”的谆谆告诫，那边又有“只工作不玩耍，聪明的孩子也变傻”的真诚建议。

在软件开发行业中，产品有很好的功能，性能却很差，这样的产品是很难有市场的；而如果一个产品性能很优越，但功能却过于简单和初级，那么这样的产品也很难赚到钱。本小节将侧重谈一谈功能好坏对于软件开发的重要意义。

“蔡佳娃，今天咱们来点深奥的话题，谈谈软件开发的目標。”

“这么官方的话题啊，那是我们这些人研究的吗？”

“怎么不是啊，这正是我们这些开发人员需要研究的呢。只有很好地实现了功能和性能两个目标，开发出来的产品才有卖点，要不然扔在大街上，免费提供甚至捆绑下载都不会有用户的。”

“是吗？那师兄你还是赶紧把这些深奥的知识传授给我吧。”

“呵呵，我们先来谈谈功能对于软件开发的重要意义吧。”

功能是一款软件产品的门面，是软件的使用者最先想到和最直接关心的部分。功能很重要，一款软件如果没有功能，也就没有商业价值了，更不要去谈其性能如何了。

尽管功能并不是万能的，不足以独立带领一款软件走向成功，但功能是底线，是一款软件在市场上存活的必要条件。所以身为开发人员，在工作中一定要保证所开发的项目能够准确无误地实现当初制定的目标功能，这样至少不会让自己的成果被市场遗弃。

“蔡佳娃，我们来举个例子，说明一下功能作为一款软件立足之本的重要性。”

“嗯，是不是师兄你的光辉历史啊？”

“呵呵，不算什么光辉历史，当时我所在的公司主要做的是建筑预算，我之前也跟你提到过，当时还有一家公司也在做这方面的开发，不过他们的技术水平不行，于是他们走低端路线，做出来的软件功能少，但是价格也相应便宜许多。”

“那你们公司呢？”

“我们公司比较务实，开发出来的软件产品功能比他们要强大，用户在使用我们的软件时还可以进行二次开发，即开发特定的预算子项。二次开发功能大大提高了我们软件的适应性与灵活性。”

“那最后应该是你们公司赢了吧。”

“其实我们的软件比他们的要贵很多，但是高端的功能摆在那。你要明白，功能上去了，客户很少会因为钱的关系而委屈自己使用二等软件的。”

一款软件产品要想做大做强，在其功能中就必须要有能够吸引眼球的地方。做软件开发与摆地摊在这方面没有本质区别，没有吸引人的商品是很难做成生意的。

比如现在提供互联网服务很赚钱，便想去做个新闻发布网站之类的东西，尝尝一夜暴富的感觉。可是真正操持起来，却还是那些浏览器和 Web 服务器传统的请求-响应模式。这样做出来的 Web 应用，如何跟那些通过广泛采用 AJAX 等前沿技术提供新的功能体验的如 Google 一般的网站 PK 呢？

“蔡佳娃，我想强调的是，在一个软件系统的开发过程中，不仅要保证实现功能，而且还必须是能流行起来、一呼百应的功能；不仅要避免开发那些老掉牙的功能，还要杜绝盲目跟风、开发一些其他人都已掌握的功能。”

“不要开发老掉牙的东西我懂，其他人都有的功能为什么不能开发啊？”

“举个例子来说吧，现在的网络游戏你玩不玩啊？”

“很早就已经不玩了。一个是因为工作忙，另外就是至今也没发现一个好玩的游戏。”

“这就对了，看看现在的网游市场，尤其是 Web 游戏非常热门，但是试着选一两种玩玩，发现每种 Web 游戏都是千篇一律、换汤不换药的一种娱乐模式。”

“对啊，现在的网游是一天就有七八个强势公测、火爆注册的，就是没见几个真正好玩的出来。”

“所以说这种在功能上无创新、没有特色的软件，一开发出来就立刻淹没于众多的孪生兄弟姐妹中去了，泯然众人矣。要想通过开发软件走向成功，必须让自己软件的功能胜人一筹。”

#### 7.4.2 提升性能带来质的飞跃

前文中也提到了，单纯靠功能是无法带领一个软件产品走向成功的，在这里制约一款软件成败的另一个因素，就是其执行性能。如果说新奇的功能让一款软件产品走上了市场的舞台，那么要想在舞台上站稳脚跟，就要靠软件产品的性能了。

“谈罢功能，我们现在来研究研究性能对于一款软件产品成败的决定性作用。”

“说实话，师兄，我对性能这个问题目前还真没什么概念。”

“慢慢地性能就会引起你的重视了。从我们上大学时起，对于在校的编程体验，也只是停留在将题目计算出正确结果这个层次，很少去研究如何提高性能。”

“对啊，那个时代只是比比谁能做出来，几乎没人会扯到运行速度上的。”

“可是到了软件开发这个行业，性能是一个经常被忽略的重要因素。”

关于性能在软件开发行业中经常被忽略的原因，可以总结为如下两条。

- 客户没说不代表不需要

很多时候功能受到的关注太多，因为一款软件产品的功能就像秃子头上的虱子——明摆着的事，所以很多客户在与项目开发方讨论项目的时候，对于性能一般都很少过问或直接不作要求。

但是如果认为只做好功能就行，不用关心性能，那就大错特错了。用户没提项目的性能问题，也许是用户不懂，没法说明白，又或者是用户已经在心中默认了一个性能标准。等到验收的时候，客户可是绝对不会手软的，性能达不到使用的要求，客户绝对不会痛快地付款。所以平日里开发功能的时候，一定要关心一下性能，防止自己到了客户冷脸的那天吐血。记住一句话：性能是隐含的刚性需求。

- 从单机时代到网络时代

“另一个让人们忽视性能的原因就是人们忽略了市场对于性能要求的转变。”

“对性能的要求还有过转变吗？”

“是啊，一开始的时候，计算机都是服务于一个人的，软件基本上都是单机版，当时能有个计算机使就很不错了，基本上没人抱怨的。”

“是啊，那个时候人们是很知足常乐的。”

“其实，当时计算机运行速度的差别，也是很难感受得到的，比如对于一个人来说，很难将小于 200ms 的时间间隔区别开来，也就是说 10ms 和 2ms 对于自然人来说是一样快的。”

其实，在早期的计算机软件开发中，由于计算机硬件的限制，开发人员在编程中会自觉自愿自发地注重性能，千方百计地优化代码，否则软件根本无法正确运行。因此可以说，早期的软件开发中其实对性能的要求是很高的，但由于硬件性能差一些不觉得而已。

后来随着单机软件技术越来越成熟，同时硬件大幅度的提速，尤其是英特尔推出奔腾系列处理器后，计算机的硬件水平已经远远高于软件的发展水平。于是，在单机软件的末期，计算机硬件十分大度地包容了软件技术对性能的要求。在这个时期，很多软件开发人员都不太注重性能方面了。

“师兄，那后来呢？性能到底是什么时候开始被重视的啊？”

“单机软件时期过后，快速发展的互联网技术将我们带进了网络时代。随后，一直低调示人的性能开始唱起高调，并越来越被人们所重视。”

“具体点说，性能是如何从幕后走向台前的啊？”

“随着网络技术的广泛应用，便出现了很多基于网络的 Web 站点或是企业级应用，而这些都是基于服务器为客户服务的。由于是服务器，需要有并发的能力。这么一来，单机时代不需要区分的 10ms 和 2ms 也开始变得重要起来。”

比如自己的服务器处理每个用户请求的时间为 10ms，而竞争对手只需要 2ms 就可以完成一次请求处理，那么一秒钟内，自己能承载的客户数量就是 100，而竞争对手则是 500，差了 5 倍，但是双方的硬件投入是基本相同的。



在这种情况下，如果用户还想与其竞争对手保持相当的服务能力，那么就要购入 5 倍于竞争对手的硬件设备并支付 5 倍于竞争对手的电费等费用。如果把自己的机器托管在电信机房，那就要再多付出 5 倍于竞争对手的托管费。”（见图 7-5）

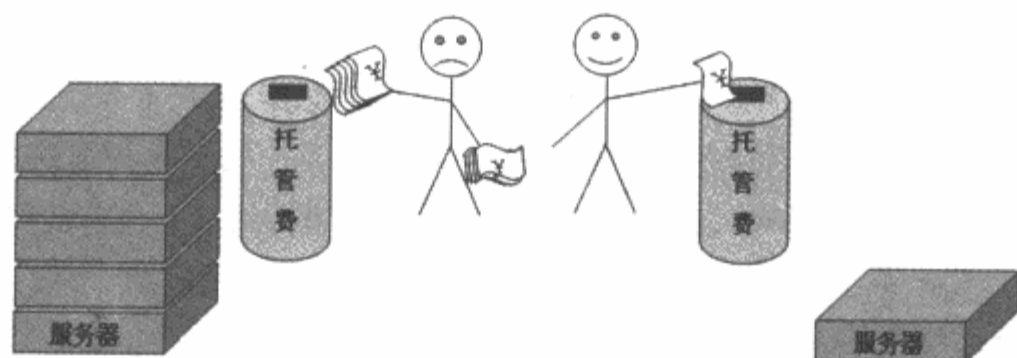


图 7-5 软件性能的高低带来的不同硬件支出

很多原因都导致了开发人员对于软件性能的忽视，这是十分错误的。其实在当今软件开发业中，不客气地讲，功能是可以拼凑搭配出来的，但是性能却是至关重要。需要高度重视的因素。

“蔡佳娃，我来给你举个例子，说明一下产品性能对于一个公司发展的影响程度。Youtube 你应该知道吧？”

“嗯，知道，是一家非常有名的视频分享网站。”

“Youtube 创立之初，也许它的老总并没有打算把 Youtube 发展成为一个世界级的视频分享网站，所以并没有过多地关注网站服务器的并发处理和分布式存储能力，就开放给用户使用了。结果 Youtube 一经推出，立刻风靡全球。”

“是啊，Youtube 算是第一个以视频分享为主的网站呢。”

“但是数天之后，由于上传和观看视频的人太多，网站的服务器瘫掉了。这可要了命了，因为如果不将网络迅速恢复，竞争对手如果趁机也提供类似的服务，那么客户就全跑掉了。”

“是啊，Youtube 既然做了第一个，就肯定会有第二个、第三个视频网站出来。”

“后来 Youtube 的后台工程师连夜修复服务器，改良架构，终于在竞争对手出招之前把用户稳稳地留在了 Youtube。”

通过上面的案例可以分析得出，Youtube 一开始的成功原因，就是其新颖的创意，给用户以全新的功能体验。而差点让 Youtube 死掉的，却是对性能的不够重视。可以肯定的是，在 Youtube 一炮走红之后，肯定有很多公司也在策划着推出第二个、第三个 Youtube。如果 Youtube 在解决性能问题上有丝毫的懈怠或迟缓，那么很可能 Youtube 就不会撑到今天而成为老大了。

一个好的创意可以诞生一颗明日之星，但是若想要把“明日”两个字去掉，则必须在性能上狠下工夫。一个公司安身立命之根本在于其软件产品的性能是否在市场上站得住脚。功能可以抄袭，但是性能却只能自己去保证。

“提到 Youtube，我们也可以看看国内的视频分享网站做得怎么样。蔡佳娃，你认为咱们国内的视频网站哪个比较好呢？”

“这个我说不好，它们基本上都是一个水平的吧，看视频的速度和清晰度都差不多。你觉得呢，师兄？”

“我是以一个客户的观点来看待这个问题的，我觉得你说得挺对的，现在的主流视频网站基本上都给人同样的流畅度和清晰度体验。在这里我要提一提我对迅雷看看的体会，迅雷看看是后来才加入到视频共享的大队伍中的，不过它依靠着迅雷平台上资源的高速下载优势，在速度上超越了很多前辈。”

“对，这点我倒是有体会，迅雷看看速度就是快。”

“是啊，而且迅雷看看上面的视频清晰度也比其他网站上的高。所以说迅雷看看能够在如雨后春笋般的视频分享网站中占有一席之地，应该归功于其对性能的把握。”

与 Youtube 不同，迅雷看看并不是以行业的先驱者走进人们视线的，甚至可以说是慢了其他人许多拍，不过迅雷看看能够与前辈们平起平坐，性能方面的优势肯定是其中一个重要的原因。

很多时候第一个把好的创意付诸实践的，未必就能够做到该领域的老大，因为功能无法申请专利，也不可以作为商业机密，一款软件产品的功能必然是公开的。好的点子就像是石子，“一石激起千层浪”，推出后马上就会有人跟风，如果扔出石子后没有在产品性能上下苦工夫，那么结局就是看着别人的浪花飞溅，自己的石头沉底了。

“现在的软件开发市场大致可以分为企业级或服务器级的软件、Web 应用开发、嵌入式开发、游戏开发等几大类。我们可以分别来研究这些分支中性能的价值。”

“嗯，那师兄你讲吧。”

“企业级开发一般面对的都是大型的软件系统，比如电子商务、电子政务等，这些应用一般都部署在服务器上，所以就像我们之前讲的那样，几毫秒的性能差异，也可以带来成本上的大不同。”

“是啊，区区一毫秒得放大成多少钱啊！”

“所以说评价大型系统的好坏大致有 3 个层次：（1）实现了要求的功能。（2）功能实现，性能在当前技术条件下达到极限。（3）功能实现，性能也完善，有好的可扩展性、可靠性及安全性。”

“哦，看来性能在评价大型系统优劣时是十分重要的啊！”

“嵌入式开发就更需要把心思用在性能上了。嵌入式设备由于体积小、用电受限，所以一般功耗都比较低，而在同等技术条件下，功耗越低，硬件性能也就越差。但是用户体验的要求却越来越高，所以在软件的性能问题上，必须慎之又慎。”

“的确，越小越费事。”

“而 Web 应用和游戏则更是用户说了算，尤其是网络游戏，现在的游戏玩家根本不会有半点迁就，只要服务器不流畅就会抱怨，除非在功能上别人无法企及，否则立马走人。”

“对啊，现在的游戏市场这么繁荣，不愁没得玩，只愁不能好好玩。”

总之，性能的优劣最后都会反映到用户的体验上来。功能对于软件开发来说比较好描绘，说白了，就是通过鼠标和键盘的操作，使显示终端上的像素点发生变化而已。而性能就很难做到完美了，性能才是最能代表一款软件甚至一家公司的核心技术水平的。

出奇制胜，应该是推出一款软件产品的最高目标，也是很多 IT 巨头的发家良策。“出奇”靠的应该是有创意的功能，而“制胜”则是依赖于强劲不含糊的性能。

## 7.5 多多参加技术大会和沙龙

7.2 节中讨论了通过读书提升自己技能和了解新技术的问题，本节将介绍另一种了解技术趋势、提高自己的途径，那就是参加技术大会。如果开发人员拿技术当饭吃，那么参加技术大会就是一场技术的饕餮盛宴了。

### 7.5.1 何为技术大会

技术大会这个概念对于一些读者来说可能并不熟悉，技术大会不是单纯地指一群人坐到一起谈谈天吃吃饭就完事的。技术大会作为技术市场上最常见的一种竞争与交流手段，对于软件厂商、开发者、合作伙伴及客户都有非常重要的意义。

“蔡佳娃，你有没有参加过技术大会啊？改天我们一起去甲骨文开发者大会啊？”

“啊，这些大会我们这些菜鸟也可以去吗？”

“当然可以去啦，你以为技术大会是什么样子的？”

“好多专家学者凑到一起，决定一下某个技术的发展方向，制定个标准或者协议呀什么的，再搞搞发明创造弄出些新技术之类的。”

“呵呵，我所说的技术大会可不是这样的。看来你对这个可以快速提高自己的途径并不是很了解啊，还是由我来给你介绍一下技术大会到底是什么东东吧。”

可以从以下几个方面来了解技术大会。

- 举办方

技术大会一般由软硬件厂商举办，这些厂商基本上都是业界首屈一指的巨头，如 IBM、微软、甲骨文、谷歌、Intel 等，大会上各厂商会大力推广自己领域的相关技术。少数的技术大会如中国软件技术大会则是由中科院软件研究所和其他机构联合举办的，这些大会主要侧重于当前行业的技术分享。

- 参与方

技术大会的参与者主要都是开发人员，在这里可以见到普通的开发人员或是学生，也可以见到在某个领域造诣颇高的技术大师。收费大会的参加流程一般是网上注册报名，随后在指定日期之前支付门票费用；而免费大会就只是在网上注册，先到先得。

- 活动内容

技术大会的活动内容有技术讲座和演讲以及新技术体验，主要的活动形式是讲座和演讲。一般技术大会的规模都比较大，在主会场进行完开幕式后会分成几个分会场来进行具体的活动。同一时间不同分会场讲座的主题不一样，同一个分会场不同时刻讲座的主题也不一样。所以在参加技术大会的时候，就需要根据自己的爱好进行取舍，规划自己的行程了。

- 价格

价格大概是每个读者最关心的问题，技术大会一般都是收费的，而且价格不菲，门票价格大概在几百元到 1000 多或者 2000 元左右。这样也将大会的人数限定在一个合理的规模上，同时收费的技术大会也就保证了提供的技术和体验是最有价值的。能够保证物有所值，甚至是物超所值。

不过，市场上也有很多免费的技术大会，比如谷歌开发者日技术大会、中国 Ruby 大会等目前就是免费的，但不知道这些技术大会未来会不会有收费的打算。

### 7.5.2 我们为什么去技术大会

“哦，原来技术大会是这个样子的啊！那师兄，开技术大会的目的是什么呢？”

“不同的人来参加技术大会目的肯定是不同的啊，首先说说身为主办方的软硬件厂商，他们举办技术大会的目的在于推广本公司的平台或技术。推广的途径就是邀请一些业内高人介绍技术的发展趋势和新技术展望，并以此来吸引开发者，扩大自己的产品在市场上的占有率。”

“既然如此，那我们这些开发者过去干什么啊？”

“去参加当然有好处啦，参加技术大会可以长很多见识呢。”

虽然大部分厂商举办技术大会的动机就是让自己的产品更加好卖，但是会议的主题还是以技术为主。因此对于开发人员来说，技术大会的重要性不亚于读一本好书，甚至效果会比读一本好书还要好。开发人员参加技术大会可以收获的内容主要有如下几个方面。

- 基于当前技术的新解决方案

对于当前流行技术的最新讲述，展示用其解决问题的新思路、新方法。同时也会介绍当下技术的最新动态，让开发人员及时掌握最新的信息。

- 新技术的介绍

技术大会中最激动人心的莫过于对新技术的介绍了。通过了解那些现在未曾广泛应用的新技术，对于开发人员制定自己的发展方向很有帮助。

- 树立自己的奋斗目标

对于我们这样的开发者来说，没有比见到业内的高人更能激励人奋进了。听一听或许是自己偶像的高人的讲座，见识一下其他人的优秀与自信，展望一下最新技术的发展，一定会让自己的心潮澎湃，斗志昂扬，快速找到自己的前进目标。

- 新产品的了解

在很多技术大会上，主办厂商都会发布自己的新产品。大会的参与者可以亲身体验这些新的产品，同时还可以与这些新产品的开发人员讨论。

- 与专家面对面

大会上一般会有很多专家的演讲，如 Java 之父、Struts 之父等。演讲后一般都有交流的时间，通过与专家交流可以了解很多新的知识或解决自己工作中遇到的很难解决的具体问题。

### 7.5.3 技术大会 PK

现在市面上存在的技术大会，除了有像 IBM 开发者大会、微软技术大会、Sun 开发者技术日大会、甲骨文开发者大会等传统老字号技术大会，还有如 Google 开发者日大会、微软中国.NET 技术大会、中国 Ruby 大会等年轻的技术大会。除了有像诺基亚论坛技术日等专业领域狭窄的技术大会，也有像中国软件技术大会这样综合性的大会。不同技术大会的认可程度统计如图 7-6 所示。

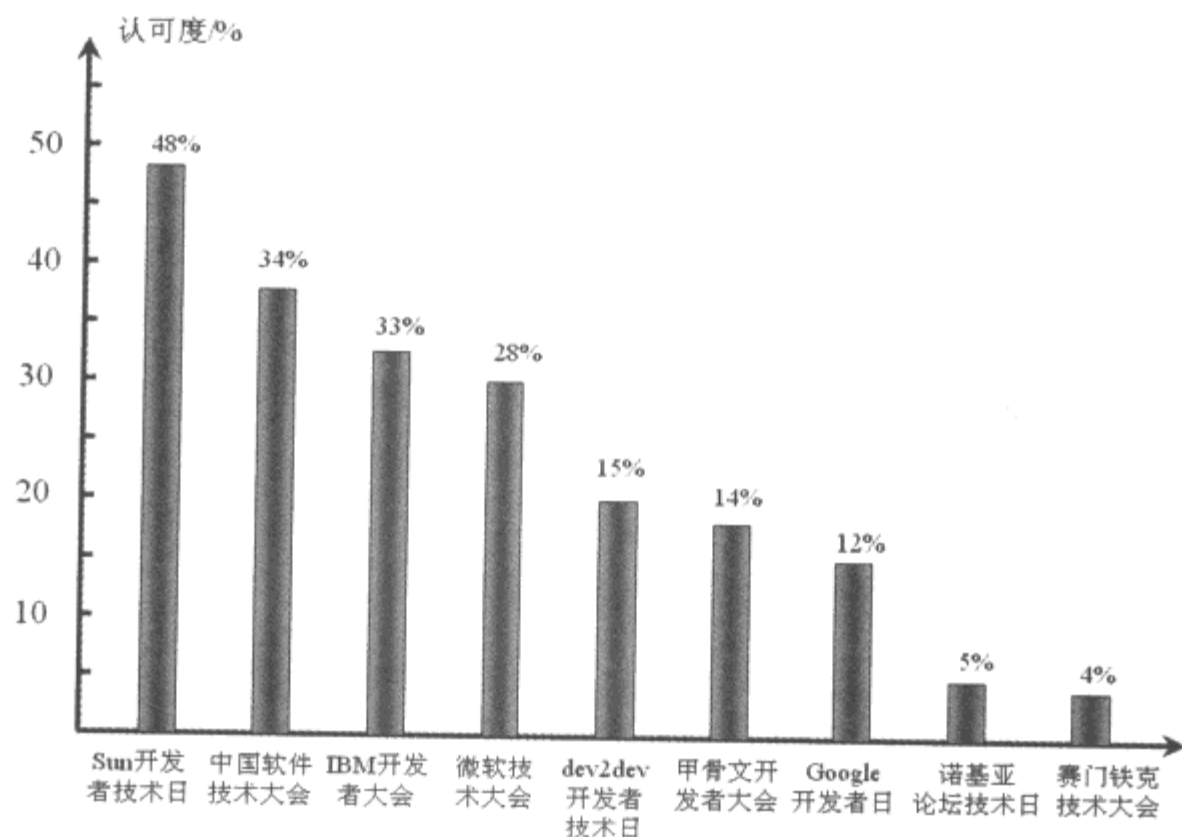


图 7-6 技术大会认可度统计图

图 7-6 统计的是参会者对各个技术大会的满意比例，读者可以从该图中看出不同的技术大会在开发者参与后的满意程度。其实还有一个特征可以用来对技术大会进行排名，那就是影响力，不同技术大会对于 IT 行业的影响力统计如图 7-7 所示。

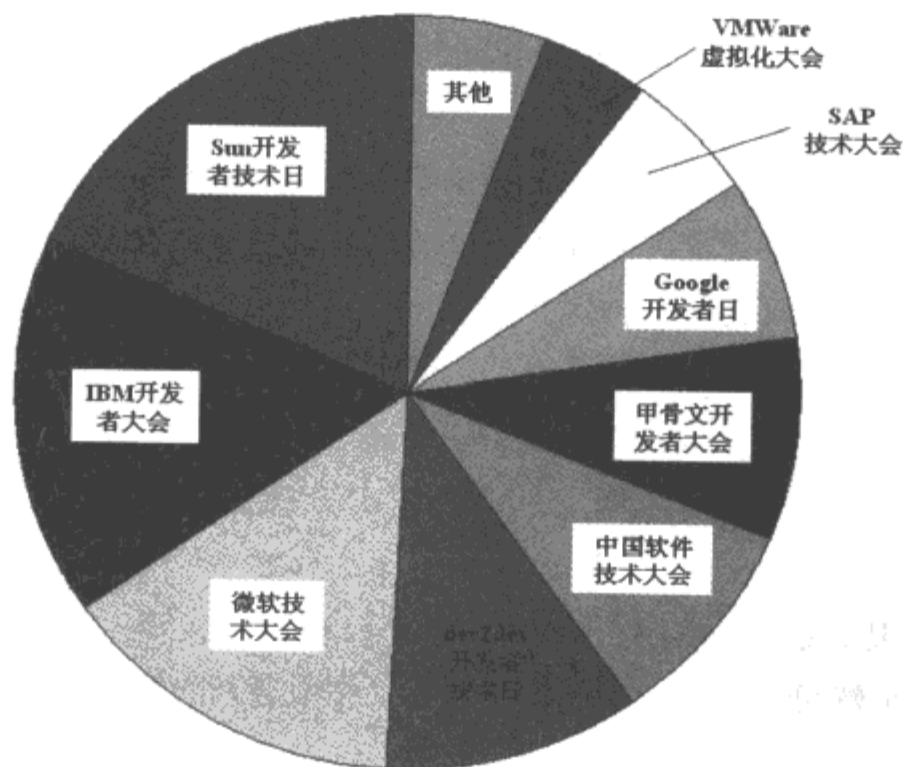


图 7-7 技术大会影响力统计图

**提示** 不同机构的统计数据可能会略有不同，但宏观上业界的情况是基本一致的。

下面，本书将选择几种影响力比较大的技术大会为读者朋友做一个简单的介绍。

### 1. Sun 开发者技术日

Sun 公司的开发者技术日是目前流行度最高、在业界最有影响力的技术大会之一，已经举办



了数届。Sun 开发者技术日的火爆与其拥有广大的开发者群体是分不开的，Sun 的 Logo 如图 7-8 所示。



图 7-8 Sun 的 Logo

Sun 开发者大会上主要演示 Sun 公司正在研究的新技术或新产品，参会者可以先睹为快。主要的介绍方向为 Java 技术和 Solaris 技术，其中以 Java 技术为主。如 2009 年 7 月份的 Sun 开发者技术日就主要介绍了 Java EE Web 平台的新特性、Java FX 和 SOA 的发展趋势、Open Solaris 操作系统、嵌入式平台 phoneME 等。

**提示** Sun 已于 2009 年被甲骨文收购，因此以后想听到相关内容的读者只能去甲骨文大会了，这在后面会有所介绍。

## 2. 中国软件技术大会

与其他的技术大会不同，中国软件技术大会是一个中国人自己的软件技术交流平台。而且作为第一个也是仅有的一个不是由软件厂商主办的技术大会，中国软件技术大会真正做到了技术的公平。中国软件技术大会的 Logo 如图 7-9 所示。



图 7-9 中国软件技术大会的 Logo

作为一个综合性的技术大会，中国软件技术大会包含的内容很广泛，从新技术新产品的演示和体验，到真实案例的讲述和行业客户的交流。除了会研究软件开发的技术和方法，也会讨论软件工程的管理科学，进行不同软件系统平台的交流。

## 3. IBM 开发者大会

IBM 应该是众多开发者心中最为向往的地方之一了，单单是这个品牌打出来，也会有很多人追随。IBM 的 Logo 如图 7-10 所示。在 IBM 开发者大会中 IBM 也会介绍很多其正在研究的新技术或新产品，当然也包括 Java。



图 7-10 IBM 的 Logo

其实 IBM 一直很热衷于推广 Java 技术，它的企业级解决方案与平台基本上也是基于 Java 技术的。由于 IBM 在业界的高大形象以及对 Java 的热衷程度，一度有很多菜鸟以为 Java 是 IBM 发明的。

#### 4. 微软技术大会

微软技术大会，又称 Tech·Ed，是微软公司一年中在中国举行的规模最大的一次技术大会。微软的 Logo 如图 7-11 所示。



图 7-11 微软 Logo

Tech·Ed 在为参会者提供来自微软专业技术人员和业内专家的讲座的同时，还会提供一些动手实验的活动。对于微软阵营的开发人员来说，这可是一场不可多得的技术盛宴了。在几天的会议中会推出上百场的技术演讲，参与者面临的主要问题是如何在各个主题之间取舍。

#### 5. dev2dev 开发者技术日

dev2dev 开发者技术日由 BEA 公司主办，其 Logo 如图 7-12 所示。BEA 公司是 Java 中间件市场中的主导者之一，其推出的 WebLogic、AquaLogic 等平台被很多电信、银行等大型机构采用。dev2dev 开发者技术日每年举办一次，旨在让广大开发者了解当前企业级应用的潮流与技术。最让广大开发人员高兴的是，dev2dev 开发者技术日是全免费的。



图 7-12 BEA 的 Logo

**提示** 很遗憾，BEA 公司已经被甲骨文收购，广大开发人员喜爱的 dev2dev 开发者技术日也就黄鹤一去不复返了。看来以后想听到相关技术内容只能去甲骨文大会了，甲骨文大会可不是免费的哦。

#### 6. 甲骨文开发者大会

原来甲骨文是全球最大的数据库软件公司，同时也是全球第二大软件公司。在收购了 Sun 和 BEA 之后 IT 界的红色巨人也就应运而生了，甲骨文的 Logo 如图 7-13 所示。作者估计完成收购后的甲骨文大会会比以前更加精彩，有更丰富的内容，希望新的甲骨文大会能向广大参会者奉上前所未有的饕餮技术盛宴。



图 7-13 甲骨文 Logo

#### 7. Google 开发者日

对于很多读者来说，谷歌就是网络的代名词，也是很多开发人员梦想的地方之一，其 Logo

如图 7-14 所示。谷歌毋庸置疑是成长最快的互联网公司，在运用新技术方面一直是走在前沿的。谷歌的大会目前是免费的，大会中的很多主题都是关于如何开发出丰富多彩的 Web 应用以及开源手机平台 Android。



图 7-14 谷歌 Logo

以上的这些技术大会都是认可度和影响力排名靠前的，对于一个有志向的开发者，能够参与到其中一个或几个技术大会中，对自己绝对是一次不小的激励。除了这些主流的技术大会，还有一些颇有影响力和支持者的技术大会，如下所示。

- 诺基亚论坛技术日
- SAP 技术大会
- VMWare 虚拟化大会

这些技术大会的覆盖面相对较窄，没有前面那些覆盖面宽，但它们在各自的领域中也都是领头羊，有兴趣和需要的读者也可以参与其中。

#### 7.5.4 技术沙龙

技术沙龙是技术大会之外的另一种技术交流的形式，主要的目的也是对技术的发展进行探讨和展望，只是技术沙龙比较分散，不集中。技术沙龙和技术大会的区别主要在以下几个方面。

- 免费，规模小

技术沙龙一般由某个厂商赞助或支持，如 BEA User Group、Solaris User Group 等。对于参加人员是免费的，而且一般技术沙龙的规模较小，通常只有几十人到一百人左右。

- 主题单一

不像技术大会每次召开都有很多不同主题的讲座或讨论，沙龙一般只有一个主题。而且仅是与自己沙龙的性质有关的，比如 BEA User Group 的沙龙就只会探讨 BEA 的某项相关技术。

- 更多的交流

参加技术大会一般的形式是听讲座、演讲，讨论交流所占的比例很小，而技术沙龙则会将 40% 或者更多的时间都用来进行讨论交流。

### 7.6 本章小结

本章在“百尺竿头”的程度之上，向读者介绍了几个开发人员在工作中应当注意学习和重视的地方，以及一些开拓自己眼界的途径。这些方面对于一个开发人员水平的提高有着重要的意义，同时也是走向牛人的道路上所必须经历的。

# 第 8 章 江湖多歧路

每一个豪情万丈步入 IT 职场的人肯定都想着要在江湖中功成名就，扬名立万，或者至少是安身立命，免于颠沛流离。但是江湖多歧路，与武侠小说中旁门左道或许更能出奇兵险招不同，IT 江湖中的歪门邪道对于一名开发人员的发展可是非常不利的，甚至是致命的。因此，在 IT 职场中闯荡必须要小心不要让自己走上歧路，免得前功尽弃，一无所成。

## 8.1 “学院”派和“企业”派

武林江湖中帮派林立，IT 江湖中也有门派之分，只不过不同于武林众帮派间的相安无事，各自发展，IT 江湖中的门派很多都是对立的。本节要介绍的，就是两个行事及思考方式不同的门派：“学院”派和“企业”派。身为一名开发人员，必须在这个问题上选择好自己的道路。

需要特别指出的是学院派和企业派并非以地域划分，在学校的就是“学院”派，在职场的就是“企业”派。“学院”派和“企业”派是互相渗透的，每个阵地都会有二者共同存在。只是“学院”派在学校中稍微多一些，而“企业”派则更多地分布在企业中而已。

### 8.1.1 何为“学院”派

之所以“学院”派在学校中的人数比在 IT 职场多，就是因为在学校中“学院”派要更容易生存下去。“学院”派不是本书所提倡的修身练武的作风，笔者也希望读者不要做一个“学院”派。关于学院派，本小节将讨论的有如下两个方面。

- 学院派的定义
- 学院派的表现

#### 1. 学院派的定义

“师兄啊，咱们 IT 行业是不是也分这个派那个帮的啊？今天好像听到同事谈什么‘企业’派之流的。”

“是啊，这也很正常嘛。文艺界不是可以分为偶像派、实力派和偶像加实力派等好多派别嘛，但是他们对外还不是一致都叫文艺圈啊。我们 IT 界对外也是一个整体，但是内部就不可避免地分成一些不同的门派了。”

“哦，是吗？跟武侠小说似的，听着挺有意思的嘛，师兄你给我介绍介绍吧。”

“哎，哪有武侠小说那样精彩啊，不过谈谈对你也有好处。首先说说你今天听到的‘企业’派，谈到‘企业’派，就不得不提‘学院’派了。”

“‘学院’派？就是生活在学校或科研单位的学者们吗？”

“不对，哎，什么时候你才能给我一次满意的回答啊！”

“学院”派是指这样的一类人：很关注纯理论研究，且只在乎理论上的可行性。错误地把任何

理论研究的成果都仅仅部署在自己大脑中进行运行和调试。学院派的人大多善于纸上谈兵，有时候研究的问题或理论并没有很大的实用价值。

“学院派”比较类似于唯心主义，我思故我在。认为把理论基础构造出来了，实际情况也会理所当然地按照理论上的设想发展。殊不知客观事物是不以自己的理论为转移的，理论应该以指导实践为中心，以能够成功应用为准则进行研究。

## 2. 学院派的表现

“师兄，你刚才给我描述了一下‘学院’派，那他们在职场或是学校中的表现是什么呢？总不会每个‘学院’派的人出门都举个着‘学院派’的牌子吧。”

“呵呵，当然不会，下面我们来谈谈‘学院’派的表现吧。”

‘学院’派的表现，总结一下大致有如下三种。

### ● 理论“清晰”，实践模糊

“学院”派只注重表面的纯理论研究和推导，很少动手实践验证，造成的结果是自己的理论很多时候都不靠谱。或者仅仅埋着头研究，根本不知道自己钻研的方向早已在风云变幻的 IT 技术圈中物是人非。技术过时倒还能勉强接受，甚至有些情况下得到的是错误的结论，那就很有危害性了。

“我给你讲个笑话，你就明白这种纯表面理论研究、生搬硬套的做法有多么糟糕了。”

“嗯，师兄你太有才了，笑话都这么多。”

“从前有个财主，花重金请了个先生教自己儿子读书识字。第一天财主儿子学会了‘一’字怎么写，第二天学会了‘二’，第三天学会了‘三’。财主儿子心想，原来读书识字也就这么回事啊，一就是一条线，二就是两条线。于是他就跟自己的父亲说自己已经学成了，财主听了很高兴。

第二天一早，财主把儿子叫过来，让他给远方的一个姓万的朋友写封信，告诉他马上过来一趟。财主儿子听完后便回去写信。可是到了傍晚，也不见儿子把写好的信拿过来，于是便过去看看。

结果财主发现他儿子正在一张大纸上面密密麻麻地画线，见到财主后还没好气地说：“你说你这个朋友姓什么不好，偏姓万。一天下来手都酸了，还差几百个线没画够呢。”

人的思想世界是无穷无尽的，所以“学院”派的人思考的多了，大多变得眼高手低，他们最喜欢做的事就是举一反三，最擅长的事就是生硬地以此类推。

### ● 实践甚浅，点到为止

其实“学院”派的人有时也是会实干的，只不过就像思考一样，也是不彻底的，总是喜欢套上自己的理论去做事。“很多‘学院’派的人抱着自己的研究成果其乐融融，却不知道那些知识已经被飞速发展的历史淘汰，或者已经是个沿袭多年的错误了。”

“哈哈，这个财主的儿子可真是太愚钝了。”

“你还别笑，没准你就被人灌输过这种思想呢。”

“不是吧。师兄你可别吓我啊。”

“‘学院’派在学校中可是为数不少哦，难免会让你遇上的。举个例子来说吧，在学校是不是学过二叉树的递归遍历和非递归遍历啊？”



“嗯，记得。数据结构是一门很重要的学科，我当年学得很认真的。”

“树的遍历也是一种很重要而且应用广泛的技术，可是有一些‘学院’派的老师在这方面并没有下工夫研究，对学生也不负责任。讲到递归遍历的时候，先煞有介事地发扬‘企业’派作风，为大家演示了一个先序的递归遍历程序。”（代码片段如下）

```
1 public static void preorderTraverse(MyTreeNode root){ //先序遍历的递归算法
2     if(root==null) return; //根节点为空，返回
3     System.out.print(root.data+" "); //访问节点
4     preorderTraverse(root.leftChild); //先序遍历左子树
5     preorderTraverse(root.rightChild); //先序遍历右子树
6 }
```

“对啊，这段代码是没错的呀。”

的确，上面的代码是没有错的，然后一些老师就会说：以此类推，中序和后序的递归遍历算法，只是把访问函数的位置掉个位置就行，这种说法是正确的，以此类推到这里也是对的。

不过有的“学院”派老师接下来就会说了：递归和非递归之间的区别不就是函数栈吗？那我们就再以此类推，非递归的实现只要去掉递归实现加上栈就可以了。这句话就有些不对头了，因为递归和非递归之间的转换绝对不仅是函数栈的问题，很多时候递归转为非递归不需要栈，如斐波那契数列，而有些时候就算加上了栈，也还是不够的。

“看来很多‘学院’派还是太懒惰了。”

“有的人会比较负责任，会向同学们演示一个非递归的遍历实现，但是之后也会回到那句话：以此类推，其他的非递归遍历方式只是变换一下访问函数的位置。”

“怎么？这种类推也不正确吗？”

“看来你果然只学了一二三，没学四怎么写啊，三种遍历二叉树的非递归算法都不一样。先序最简单，中序、后序复杂度依次递增。以先序为例，首先站在根节点上，将右子树入栈，把左子树作为新的根节点；根节点如果为空则弹栈，若不为空则继续将右子树入栈，左子树成为新的根节点……按照这样的规律，就可以遍历整个二叉树了。”（代码片段如下）

```
1 //先序遍历算法基本思想：站在一个节点上，打印（访问）当前节点值，当前节点右子进栈，去当前节点左子。
2 //总的是循环处理，一次处理结束后，若当前节点为 null，则不断出栈，直到当前节点不为 null 或栈空，
3 //若栈空且当前节点为 null 则遍历结束
4 public static void preorderTraverse(MyTreeNode root){
5     MyStack<MyTreeNode> ms=new MyStack<MyTreeNode>(); //创建栈对象
6     MyTreeNode curr=root;
7     while(curr!=null||!ms.isEmpty()){ //当前节点不为空且栈不为空一直处理
8         while(curr==null&&!ms.isEmpty()){ //某一节点右子树为空，继续弹栈
9             curr=ms.pop();
10        }
11        if(curr==null) break; //栈空且当前节点为 null，则遍历结束
12        System.out.print(curr.data+" "); //打印输出
13        ms.push(curr.rightChild); //当前节点右子树进栈
14        curr=curr.leftChild; //将当前节点的左子树变为当前节点
15    }
16 }
```

“嗯，师兄你的这段代码我看懂了。那中序和后序呢？”

“这是最简单的一种遍历，因为仔细研究就会发现，先序遍历中是不需要走回头路的，即每个节点只访问一遍，但是也只路过一遍，而中序和后序遍历中对于某个节点需要经过不只一次，如果你按照‘以此类推’的‘学院’派方法去做，只是改变访问函数的位置的话，你会发现你的遍历序列或者是重复打印了某个节点，或者是漏掉了某些节点。”

“哦，我好像有点开窍了。看来我果然成了财主的儿子了。”

“呵呵，中序和后序遍历的代码太长了，估计你一时半会也看不透彻，日后我再发到你的邮箱吧。不过实践出真知这句话还是要铭记于心啊。”

很多“学院”派宣称自己研究理论，但其对理论研究的态度，很多时候都像那个财主的儿子一般，浅尝辄止，以偏概全。使得自己对于理论知识没有一个整体的认知和把握，造成所探究的技术要么不符合发展趋势，要么与实际严重脱节。

“学院”派的不实践，或者是不彻底的实践，对于一名开发人员来说都是万万要不得的。身为开发人员就应该有踏破铁鞋的志气，避免仅靠直观的推论而酿成错误。

#### ● “学院”精神，著书立说

“蔡佳娃，要小心喽！不仅在学校、职场需要避免受‘学院’派的影响，看书的时候还会遇到‘学院’派的书呢。”

“啊，不是吧，‘学院’派出的书会是什么样子的呢？”

“某位文豪不是说过嘛，读一本书就像是在同作者交谈，那你想象一下和一个‘学院’派交谈的样子不就是看‘学院’派书的样子了？呵呵，‘学院’派的书大都实用起来有些困难，而且有的时候并不能把握技术的发展趋势，讲了一堆已经过时不用的东西。”

“对啊，想想他们的思考模式和开发习惯，还是不要让自己荒废掉苦苦积累的技能为好。”

“是啊，我们之前说过的那个介绍 Java 中构造器继承的书，搞不好就是某位‘学院’派高人所作呢，呵呵。所以在选书的时候要凝神啊。”

如今市面上琳琅满目的技术书籍中，也是有着“学院”派和“企业”派之分的，“学院”派的技术书籍秉承了其一贯的“重理论、轻实践”的风格，虽然一本书讲理论谈思想不为过，但是没有经过实践验证的理论和思想，对于务实的开发人员来说，还是不看为妙。

### 8.1.2 “企业”派的实干

前面也已经提到，“学院”派并不是一点实干都没有，只是他们信奉没有实践基础的理论而不是实实在在彻底的实践验证。这种错误的信仰也使得“学院”派的动手实践变成表面上的点到为止，而不去进行深入的发掘。

相比较来说，“企业”派的实干精神所带来的优势就非常明显了，“企业”派最大的特点就是理论联系实际。不像“学院”派，“企业”派的人更乐于通过实践为自己答疑解惑，而不是像“学院”派那样闭门研究。

“蔡佳娃，下面我们来谈谈‘企业’派对于‘学院’派的优势。那就是‘学院’派在少有的实践中，不注意代码的性能优劣和可维护性，而“企业”派却视之如命。”

“性能优劣我知道，可是可维护性是个什么概念呢？”

“性能优劣的问题我们已经探讨过了，这里主要说说代码的可维护性问题。可维护性说白了，就是你用代码编写的软件部署到生产环境以后，对代码进行修改时的难易程度，一个软件的可维护性也是每个开发人员都需要自觉加强的。”

“哦，可维护性原来也是需要联系实际的啊。”

“举个例子来说吧，你上学的时候写过表达式求值的程序吧？”

“嗯，写过，就是在数据结构的栈和队列那章学的。不过我忘记我是怎么写的了。嘿嘿，那时候没注意积累自己的小仓库。”

“表达式求值的一个环节就是判断两个运算符的优先级高低，这段判断运算符优先级的代码如果这样写，你觉得怎样？”（代码片段如下）

```
1 public int compareOptr(char optr1,char optr2){
//函数返回 0 表示优先级相等，1 表示高，-1 表示低
2     if(optr1 == '+' || optr1 == '-') { //当第一个运算符是 '+' 或 '-' 时
3         switch(optr2) { //分情况讨论第二个运算符
4             case '+':
5             case '-': return 0;
6             .....
7         }
8     }
9     else if(optr == '*' || optr == '/') //当第一个运算符是 '*' 或 '/' 时
10    .....
11 }
```

“我觉得也没什么问题啊，考虑得很全面啊，switch 语句写得还蛮精练的。”

不少开发人员在工作中的这种“一个解万岁，多一个白费”的“学院”派风格就是为什么代码的可维护性会如此不受重视的原因。想想看，如果需要再添加一个运算符怎么办？如果在这个代码的基础之上进行维护的话，那么就需要在后面再加一个 if 分支，同时，原先存在的 if 或 switch 语句中还要加入新的分支。这就复杂很多了。

读者朋友们可以算一笔账，设有  $n$  个运算符，对于每一次的两个运算符比较，需要考虑  $n^2$  种情况，而添加一种运算符，就会变成  $(n+1)^2$  种情况，也就意味着你需要做  $2n+1$  种新情况的添加，当  $n$  越来越大时，所需要做的工作将越来越烦琐。而且这还只是添加，如果删除运算符呢？或者需要修改某个运算符的优先级呢？岂不更让人头疼？”

“是啊，那可维护性的代码是什么样的呢？”

“根据本题，可维护性好的代码可以这样来设计：编写一个方法，接收的参数为运算符，返回表示该运算符优先级的整形值。这样在比较优先级的函数体中调用这个方法，获得所比较的两个运算符的优先级，然后只需要对整形变量进行比较即可。”（代码片段如下）

```
1 public int compareOptr(char optr1,char optr2){//比较优先级函数
2     return getPriority(optr1) - getPriority(optr2);
3 }
4 public int getPriority(char optr){//优先级获得函数，返回优先值
5     switch(optr){//根据不同的运算符返回不同的优先值
```

```

6         case '+':return 1;
7         case '-':return 1;
8         .....
9     }
10 }

```

“果然很精练啊，这样一来代码的可读性也大大增强了，如果需要增加新的运算符，只需要简单改动 getPriority 方法就行了！”

“是的。蔡佳娃，你不要以为只有学生会犯这种错，职场中也存在着‘学院’派的爪牙呢。我曾经带过的一个下属，就是这种人呢。”

“啊？那他编出了怎样‘学院’风格的代码啊？”

“当时我让他去开发一个统计用户选项的桌面程序模块，统计的内容我不便多说，姑且用业余爱好来替代吧，需要统计的选项比较多，不过方式很简单，就是接收用户的所有选项并输出即可。大概要实现的效果就是这样。”（如图 8-1、图 8-2 所示）

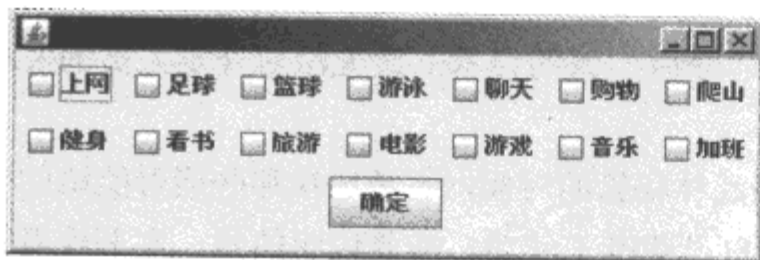


图 8-1 统计界面效果图 a

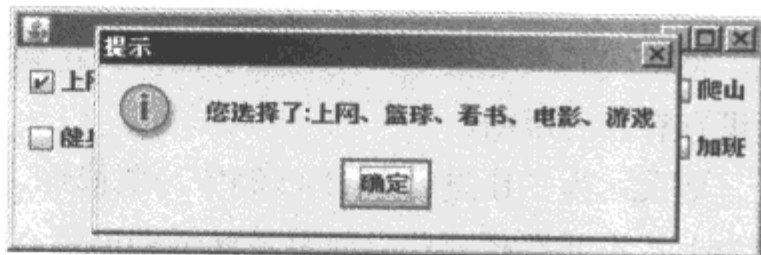


图 8-2 统计界面效果图 b

这个桌面小程序，对于每个读者朋友来说，只要稍微有一些 Swing 编程的知识，都不算难。构建界面的时候，只需要创建一个窗体类 JFrame 类的对象，将所有的 JCheckBox 控件和确定按钮添加到窗体中显示就行了。

然后在用户单击了确定按钮之后，将被选中的一个或几个 JCheckBox 控件的文本添加到输出字符串里面，在对话框中输出即可。

“哦，那应该很简单，就用 JCheckBox 控件就可以了。”

“是啊，我也认为不会有什么问题，他也够利索，很快就拿给我看，我觉得还不错，表扬了他一下。几天后，客户过来了，说要在那个模块中添加几个选项，我想这没问题，就让那个人把客户说的东西加上，那时候已经是项目临近验收的最后测试阶段了，我让他十分钟后给我。”

“然后呢？”

“他当场就面露难色，说十分钟弄不好，他得加加班才能弄好。我说怎么会改这么长时间呢？我让他把源代码拿给我看看，结果一看我呆了。”

“那是怎么个情况啊？”

“他的各个 JCheckBox 控件全部都是独立声明和创建的，这样的话再往面板里添加的时候需要一个一个地添加，用户单击确定按钮的时候也需要一个一个地判断。40 几个控件啊，黑压压一大片代码，看得真够难受的。”

“那还真挺麻烦的，尤其是添加新控件的时候，需要改动好几个地方呢。”

其实这种情况下直接用一个 JCheckBox 控件数组就行了，创建控件的时候采用枚举的方式创建一个 JCheckBox 控件数组。然后无论是添加进面板，还是用户按下确定按钮后的判断，都可以

用一个增强型 for 循环办到了。这样代码要简短很多，而且一旦选项有增减，只需要改动 JCheckBox 控件数组即可，省去了很多的麻烦。

而且，就像总说话会言多必失，做重复烦琐的工作多了，出错的几率也就会大大增加。因此，这种不考虑真正生产环境中代码可维护性的“学院”派作风，每个开发人员都要杜绝染上。

“最后，我们来谈谈‘企业’派写的书。”

“相对于‘学院’派，‘企业’派写的书往往都是多年开发的心得体会，读了非常有价值，因为归根结底，你是需要把手放到键盘上敲代码的。而‘企业’派前辈的经验之谈，无疑是职场新手最希望得到的职场‘秘籍’。”

“是啊，我也觉得‘企业’派写的书更加让人信服。”

“‘企业’派作者写的书，和现实开发世界的距离最近，所以没用的东西不会讲，而且也绝对不会让自己的书中出现模棱两可的知识，或是无法成功部署运行的代码示例。”

“学院”派的开发实战由于仅仅停留在浅层次，只是达到基本要求，而不会进行生产环境的验证，使得代码性能很差，而且难于维护。这就使得“学院”派的技术书籍无法对开发人员的发展提供有有利的帮助。

因此，为了让自己在 IT 江湖中习武修炼，选择武林秘籍的时候，还是选择那些能看能练的书籍为上，不要选择那些只能看没法练的书籍，防止自己荒废了武功。需要说明的是，本书作者是一个作风强硬的“企业”派，所以本书也是一部讲理论、重实干的作品。

### 8.1.3 一起来做“企业”派

通过前面的内容，大家可能对于“学院”派和“企业”派有了一定的了解。“企业”派跟“学院”派是大大不同的。了解了“学院”派的危害，开发人员要学着让自己做个执着踏实的“企业”派，这样才能不被 IT 江湖人笑话。

“蔡佳娃，谈了这么多‘学院’派，你大概也认识到‘学院’派的危害了吧。”

“是啊，我发现‘学院’派还真是个隐秘的大军呢，我必须要小心自己的技术堡垒不要被他们给攻陷了。”

“的确，不仅是学校的老师，同学们中有‘学院’派，职场的开发人员中也有“学院”派。有‘学院’派作风的老师会危害到自己的学生，而‘学院’派的学生如果不放下纯表面理论的大架子，专注于实践的话，便会成为一个‘学院’派的开发人员。”

“呵呵，还好不像生化危机，传染性不是很大。”

“一个‘学院’派作风的开发人员，是很难在职场中有所建树的。因此，你要努力让自己成长为一名务实谨慎的‘企业’派哦。”

其实，“学院”派也没有那么邪恶，“学院”派不是故意“学院”派的，只是他们研究知识的方式不对，对于实践和理论的结合做得太不好了。相比之下，“企业”派就很值得大家学习了。

#### ● 坚实的理论基础

“企业”派可不是光实践，他们有着坚实的理论基础，那些鼓吹“读书无用”的超实干主义者都不是真正的“企业”派。




- 实践出真知

“企业”派的开发人员遇到了问题或拿不准的地方，肯定第一个动作就是去敲代码进行验证观察，这样随后的翻书学习也会更加有效，因为“企业”派的人对眼见的事实才最信服。

- 将务实进行到底

“企业”派从不纸上谈兵，他们只有在软件成功部署到真实的硬件上并且准确无误地运行，能够满足性能与可维护性、可靠性要求的时候才会满意地点头。

理论是用来指导实践的，如何指导？只有放到实践中，才算是接受检验，检验无误后，才可以作为指导。理论联系实际是“企业”派的做事准则，希望各位读者朋友尽早抛弃“学院”派的帽子，一起来做果敢踏实的“企业”派。

 **提示** 本书中所指的“学院”派与平时学术界所说的学院派没有必然联系，只是借“学院”派的称呼特指IT行业中的一类人群而已。

## 8.2 关于“剑宗”和“气宗”的讨论

IT江湖中根据对理论和实践态度的不同，可以将开发人员分为“学院”派和“企业”派。而根据对提高修为理解的不同，可以将职场中的开发人员分为“剑宗”和“气宗”。本节将与各位读者朋友探究一下在江湖打拼中对于“剑宗”和“气宗”的取舍。

“剑宗”和“气宗”，原指金庸先生笔下华山派中的剑气两派，一个主张以剑招制敌，一个主张“练剑者以气为本”。练功的方式不同，收到的成效也不同，对应到IT江湖，只练“剑”不行，必须要修“气”，这样才可以让自己打遍天下不挨揍。

### 8.2.1 何为“剑宗”

武林中人练功常说一句话：“外练筋骨皮，内练一口气”。这里的筋骨皮，与本小节要讨论的“剑宗”指的是同一个概念。“剑宗”，顾名思义就是以剑为宗，“剑宗”派的规模在IT江湖中不在少数，下面就向读者揭秘“剑宗”派到底是以什么样的方式治学为武的。

“蔡佳娃，现在我们来谈谈江湖中的另外两个截然不同的门派：‘剑宗’和‘气宗’。”

“这不是《笑傲江湖》中的华山派吗？”

“的确，对应到我们IT行业，这就是两条需要慎重选择的成长道路。”

“是吗？看来江湖中需要小心行走的路还真多啊，那师兄你给我说说‘剑宗’和‘气宗’都是什么样的吧！”

“我们先来谈谈‘剑宗’，‘剑宗’的人重招式而轻内力，认为剑法练到炉火纯青的地步，就可以笑傲江湖。这种做法不能说全错，不过我并不是很赞同。”

“剑宗”所追求的“剑”都是什么样的剑呢？概括起来，大概有如下三种。

- 开发工具

“剑宗”派人士比较看重开发工具，如今各种各样的集成开发环境（IDE）越来越多，功能也越来越强大，而且使用也越来越复杂。基于此，能够熟练地使用IDE方便快捷地开发出软件产品，变成了每个“剑宗”人士进入职场的首要目标。

- 编程技巧

一些巧妙的编程技巧，也是许多“剑宗”派在工作中孜孜以求的剑谱，这些编程技巧很多时候并没有太高深的理论，但却是聪明才智的一种体现。从某种程度上来说，“剑宗”派也是非常讲究实干的，却因此而忽略了一些更值得去研究的思想。

- 编程语言

“剑宗”练剑，在很大程度上都是为了将某一门编程语言彻底掌握，这是“剑宗”派在武功上登峰造极的必经之路，也是“剑宗”派最拿得出手的一把利器。

“师兄，我觉得‘剑宗’派注重招式还是比较不错的，像我这样的菜鸟，进入职场要学习的，首先也是这些招式吧。”

“‘剑宗’派当然有它的可取之处，‘剑宗’派练的招式哪个你能说没有用？‘剑宗’派的人至少是个不折不扣的实践主义者呢。”

“对啊，练剑当然算是实干了。”

“对于一名新入行的开发人员，无名无分，内力修为又不够，不赶紧寻两把大剑带在身上，怎么能够求自保、求发展呢？”

对于任何一名肯吃苦的开发人员来说，练剑是比较容易的，而且收效也快。当然了，前提是能选把好剑，否则胡练歪练更容易让自己走火入魔。

“剑宗”派不能太醉心练剑，也不能太依赖于剑，不能总想着永远扛着两把大剑闯天涯。随着武功的增加，“剑宗”应该慢慢往“气宗”那里转变。武学上讲究内外兼修，这也是很多“剑宗”派人士都忽略的东西。

## 8.2.2 何为“气宗”

对应于“剑宗”的“外练筋骨皮”，“气宗”就是“内练一口气”了。IT江湖中“气宗”所练的“气”，就是编程思想。在软件开发中，“气宗”和“剑宗”最大的区别就是对于武器的依赖程度，或者说武器占一个开发人员战斗力的百分比。

“师兄，那‘气宗’是什么样的啊？研究理论？”

“你可不要把‘气宗’派和‘学院’派给弄混了啊。这俩差的可不是一点半点。先把话说前头，‘剑宗’和‘气宗’都是理论联系实践的。”

“哦，这样啊，好，师兄请继续。”

“‘气宗’是这样一种开发人员，他们在开发中注重学习编程思想而不仅仅是编程方法，而且还会主动地去研究一些如软件设计模式、平台思想等更深层次的通用思想。”

“看来‘气宗’还真是很注重内在修为啊。”

“气宗”派的开发人员研究的是编程思想，是解决问题的统一原则和规律，这是放之四海而皆准的知识，并不局限于某个平台或某种语言。

随着内力修为的增长，“气宗”派的开发人员对于“剑”的依赖程度将越来越低，就会达到金庸先生笔下独孤求败那样的境界：“不滞于物，草木竹石，均可为剑。自此精修，渐而进于无剑胜有剑之境”，这时候就是个内功浑厚的软件开发高人了。

“哇！看来修炼内功的确对于自己今后的发展大有裨益啊！”

“是啊，不过修炼内功可没那么容易搞定，练气要比练剑难多了，这一点从“剑宗”和“气宗”的成长曲线图中可以看出。”（见图 8-3）

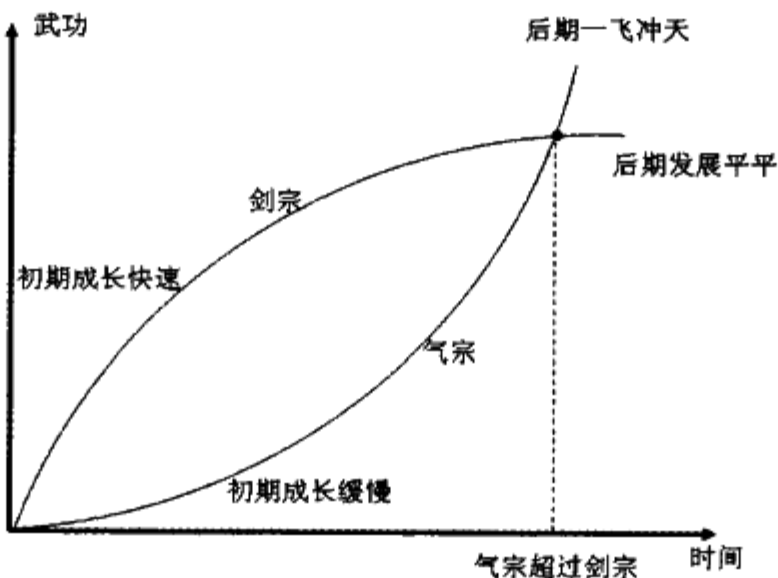


图 8-3 “剑宗”和“气宗”成长曲线图

“内功的积累是个厚积薄发的过程，不像练剑可以迅速成才，练气的收效很缓慢，直到随着修炼的加深，内力上升到一个高度之后，才会拨云见日，自己的水平也会如大鹏一般扶摇直上了。而只练剑的开发人员的能力到了一定程度后，想再发展就会很难了。”

在这里，不妨将“剑宗”和“气宗”系统地做一个对比，从图 8-3 中也可以看出，“剑宗”的优势是初期成长快，但后期很难再有所成；而“气宗”在初期颇为艰辛，但是随着修为的提升，成长速度将越来越快，最终将超越同期的“剑宗”，步入高人的殿堂。

可以将“剑宗”比喻为技术快餐，吃得迅速还管饱，但是留不住多少营养；而“气宗”更像是正餐，做得慢，但是有营养。“剑宗”就像是计算机的内存，开机状态下内存存取速度超快，当前程序运行也全在内存中进行，但是一旦关机，所有的东西就会丢失；而“气宗”则是硬盘，剑法的招式可能会因为时间或其他原因慢慢淡忘，但是学到的内功除非自己一掌废掉，否则会一直老实实在地待在硬盘里。

“蔡佳娃，你对音乐的知识了解多少啊？”

“嗯？干嘛问这个啊，我可是个音乐白痴啊，只知道那七个音符。”

“呵呵，这就够了，你有没有这样想过，区区 7 个音符，居然通过不同的排列组合以及变调，把我们带到了如此浩瀚飘渺的音乐世界。”

“是啊，听你这么一说，还挺神奇的啊。”

“是啊，这都是无数作曲家的天才创意。其实说这个无非是想说软件开发就像是谱曲，用有限的音符谱写出无数精彩的曲目，比如 Java 的类库是有限的，但是用 Java 开发出的应用却是无穷无尽的。”

“看来，两者还是有一些共性的。”

“作曲家谱曲子靠的是对音乐的感悟和天才，与之类似，我们软件工程师开发项目，靠的也得是编程的思想，即深厚的内功。所以想要让自己开发的产品创意无限，不加强内功怎么行啊。”

还需要注意的区别就是“剑宗”如果没了“剑”，比如换了编程平台或方法，那就发不出什么狠招了，而“气宗”的气则永远不会背叛自己逃走。所以多研究大道无形的东西，还是很受用的。招式可以一下就学会，但是内功不是一天就能够练成的。练气，或许才是以不变应万变的哲学。

将“剑宗”和“气宗”作对比的时候，需要说明的是IT江湖中没有单纯练剑的“剑宗”和单纯练气的“气宗”，“剑宗”虽然会把更多的精力放在“剑”上，但是他们也需要练气，要不然没有足够的内力根本使不出漂亮的招式，只能简单地挥几下不消耗内力的剑法。同理，“气宗”也需要练剑，只不过更加注重练气的“气宗”在内力的优势下，学剑会学得更快。

### 8.2.3 奇技淫巧不如提升修为

很多时候有些“剑宗”派的开发人员对于练剑过于执着，就会变得只喜欢研究奇技淫巧。这些人本来就疏于修炼，内力不足，对奇技淫巧的过分追求就更显得不恰当了。

“蔡佳娃，在你步入职场的初期，研究研究剑法是可以的，但是注意不要在奇技淫巧上面多做停留，否则对于自己的武功提升不会很有利。”

“师兄，什么是奇技淫巧啊？”

“奇技淫巧就是过于奇巧甚至于无益的东西，在开发中很多小技术都属于奇技淫巧，如一个漂亮的椭圆按钮、一个吸引人的异形窗口等。”

奇技淫巧并非没有用武之地，很多公司对于这种醉心于剑法的开发人员都有一定的需求，但是靠着奇技淫巧永远不可能走得太远。因为这种一招半式的精妙根本打不过内功深厚的如来神掌，内力修为高的开发人员不是不会奇技淫巧，而是不花很多精力去研究。若要真比起来，有内功基础的奇技淫巧或许更会让人赞叹不已，如图8-4所示。

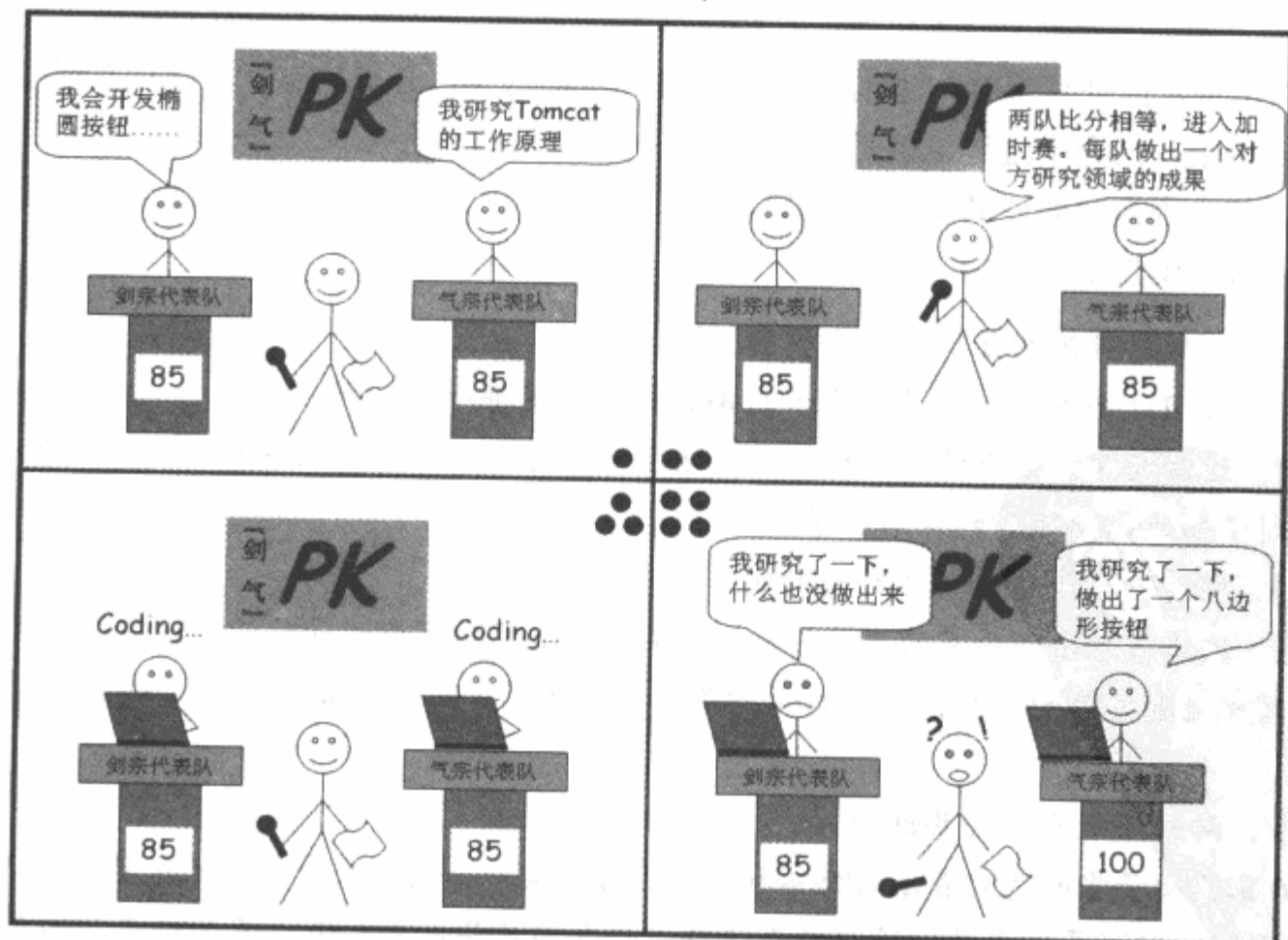


图 8-4 奇技淫巧 PK “气宗”

如图 8-4 中所示,研究奇技淫巧虽然很有成就感,能逞一时之快,但是面对那些大道无形的东西就没办法崭露头角了。由“气”入“剑”易,或许只是扫一眼代码就参透了其中的奥妙,而由“剑”入“气”难,没有长时间的参悟只能是百思不得其解。所以说尽管奇技淫巧现在仍然有市场,但任何一个目标远大的开发人员都应该停下来让自己修修内功了。

“蔡佳娃,问你个问题,你觉得屠龙刀有用否?”

“当然有用啦,宝刀屠龙,号令天下,谁敢不从?”

“这样设想一下,如果把屠龙宝刀给一个武功不高的江湖小虾,他肯定无法驾驭宝刀的霸气,挥来挥去必定要伤到自己,而能够安抚宝刀霸气的人中,又有哪个大侠愿意整天背个大刀走来走去的?高人们追求的都是‘无剑胜有剑’的境界,所以宝刀屠龙给谁都是毫无价值的。”

“听师兄你这么一说,果然是这个道理啊。”

“引申到‘剑宗’和‘气宗’上,也是这个道理,很高妙的‘剑’,菜鸟根本就掌握不了,而‘气宗’派的高人又对此不予理会,所以对剑法的研究最好是‘浅尝辄止’,不要过分投入精力研究,速速提升内力方为上策。”

身在职场中的开发人员,应该遵循着以“气”为主、以“剑”为辅的原则,学会在一招一式中去领悟其中的内功心法,进而提升自己的内力。如通过从事 Java 的软件开发,逐步去感悟面向对象编程的思想,然后用修炼得来的内力去化解遇到的问题,这才是武林宗师之路。

可以用一句篡改后的古语来作为本节“剑宗”和“气宗”之争的结论:工欲善其事,必先会舞剑,工欲成大事,必要修其气。

## 8.3 有自己的平台才是王道

软件开发平台,或许是每个开发人员都会在工作中接触到的,如今的 IT 江湖中存在着各种各样的平台。在进行本公司的项目开发时,是就着市面上现有的平台技术搭建,还是首先为自己的公司开发出一个专用的平台再进行功能开发,这个问题是非常值得探讨的。

### 8.3.1 关于框架的纯“拿来主义”

作为一名 Java 开发人员,肯定不会对开发平台感到陌生,平台是基于特定的编程技术、技术架构或业务逻辑建立起的为了实现快速设计、开发、调试和部署而存在的技术工具。采用平台技术进行软件开发,不仅能够提高效率,还可以提高项目的可维护性、可扩展性等方面的指标。

“蔡佳娃,做开发人员也这么久了,对平台技术有一定的了解了吧?”

“师弟我不才啊,一直没弄懂平台和框架的区别。”

“呵呵,这个和问题的范畴有关,一般广义的平台是包含框架的,还包括像 Tomcat、Weblogic、JBoss 等应用服务器。而狭义的平台或者指这些应用服务器,或者指专注于某个层的解决方案的框架技术。不管广义还是狭义,它们都可以避免重复开发和提高开发效率与质量。”

“哦,原来是这样啊,看来是我想得太多了。”

如果从层次的角度考虑,用户的最终应用是基于框架技术开发的,如 Struts、JSF、Spring、



Hibernate 等，而框架的正常工作又需要应用服务器的支持。实际上把框架技术部署到应用服务器上面也可以看做是对服务器功能的一种扩展。

其实业内对于框架和平台并没有明确的界限，平台既可以指应用服务器如 Tomcat 等，又可以指框架技术如 Struts 等。但不管是框架或技术，平台的根本特性是不会变的，就是把众多解决方案中具有共性的部分提取出来，优化精炼，使开发人员在开发过程中只关注不一样的地方即可。

比如 Spring 平台把对象间依赖性的管理实现了，开发人员在使用 Spring 开发业务层时就不再需要开发对象依赖性管理的代码，关注于不同具体的业务逻辑的开发即可。

“蔡佳娃，我们今天要谈的是是否要有自己的软件技术平台。”

“自己的软件技术平台？现在市面上不是已经有很多平台了吗？自己再去开发用处大吗？”

“肯定有用啊！市面上的框架都是通用型框架，并不一定是最适合自己的，开发自己的平台的好处就是以后再做项目可以基于此，会方便很多。”

“缺点就是不能单独收回成本啦，你想想，开发自己软件平台的客户是谁？”

“谁用是谁的，那就是我们自己喽。”

“所以说，开发成本肯定不能直接从某个具体的客户项目中收回，需要在软件平台开发出来成功投入使用，并基于它高效率地开发出多个新的用户项目的时候间接收回开发成本。”

开发自己的软件平台是一项磨刀不误砍柴工的工作，平台是一个公司做大以后的一种投资。比如只做一个项目的时候，如果先去开发平台，再去做具体功能，那样肯定是不划算的。

如果一个公司开发的业务集中在某个领域，或者相似的项目比较多，采用通用框架会造成重复开发和测试。若用一部分精英开发一个完美的平台，将行业中的公共部分加入到自己的平台中，免去重复开发的步骤，这样再进行具体项目的开发就比较实惠了。

规模大、技术强的公司一般都会去做自己的平台，因为开发平台的投资会在其效率提高的空间中抵消，但是对于规模比较小、技术比较弱的公司，做平台显然就有些吃力了。

“蔡佳娃，我可以给你举一个例子来说明软件平台的作用。不考虑软件公司的经济因素，我们把软件公司比作政府，把公司的技术能力比作政府的财政。也就是技术越好，相当于政府越有钱。”

“嗯，然后呢？”

“假定政府需要在间隔两座山的 A 市和 B 市之间修一条路，这时候就有两种方案了，一种就是修盘山公路（如图 8-5 所示），这种方式施工难度不高，造价也低。但是缺点显而易见，那就是往返两地的時候需要绕来绕去，而且因为是山路，还需要限速。”

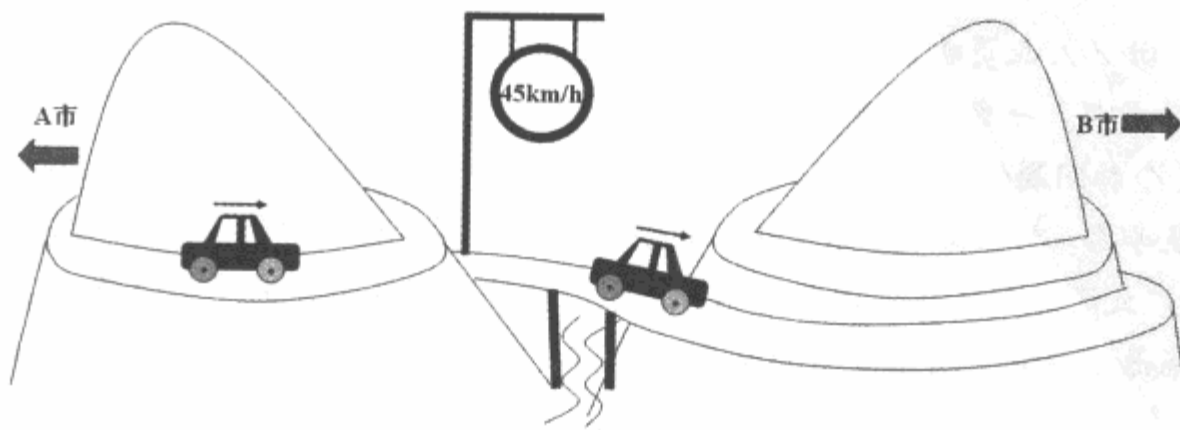


图 8-5 盘山公路式的项目开发

“的确是这样，那第二个方案呢？”

“第二个方案就是架桥、挖隧道（如图 8-6 所示）。这个方案的特点是施工难度高，造价也比盘山公路要昂贵很多，打通一座山比在它周围盘一圈公路要费钱得多。但是优点很明显，那就是地图上看是多远，现实中就是多远，还没有盘山公路那么低的限速。”

“是啊，挖隧道修的公路开起来是要快很多。”

“站在政府的立场考虑，如果一开始政府没有钱的话，肯定没办法挖隧道，但路是必须要修的，只好修盘山公路。但是慢慢政府变得有钱了，再遇到这种情况的时候，肯定就要花大价钱挖隧道修公路了，这样虽然投入高，但是带来的效益也非常明显。”

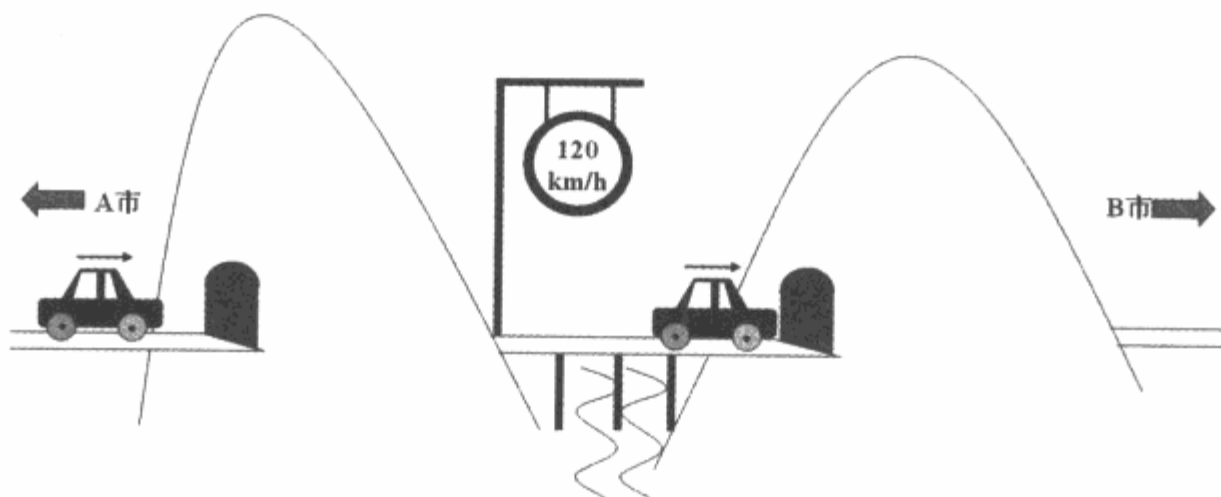


图 8-6 挖隧道式的项目开发

对应到项目开发，盘山公路的方法是公司的技术能力不够，没有能力做平台一步直达，只好利用通用的平台加上自己的开发能力进行搭建，这样虽然开车绕来绕去，但是最终也能满足项目要求，客户也能认可其造价。而如果公司发展到了规模，再不考虑挖隧道就太没战略眼光了，所以应该为了长远的打算，为自己的公司开发出一套平台来提高效率。

应该学着像国家发展一样，先修盘山公路，通过一些小项目求生存，求发展，然后逐渐通过开发自己的软件平台来建立自己的王道。

- 第一步要有自己的 Struts、JSF、Spring、Hibernate 等专注于某个层的解决方案框架的改造升级版，适合自己公司特定的业务需求情况。
- 第二步可以进一步基于 Tomcat 或 JBoss、GlassFish 等开源应用服务器打造适合自己公司特定需求的应用服务器。

以上两个步骤可以用图 8-7 来说明。第一步达成后，在自己的框架上面开发具体系统功能的时候就可以更加贴切和快速，使每个层次衔接得更为紧密。而如果能够达到第二步，那么就将这种无缝的衔接沿袭到了框架和应用服务器上面，这样开发出来的系统将更加完善。

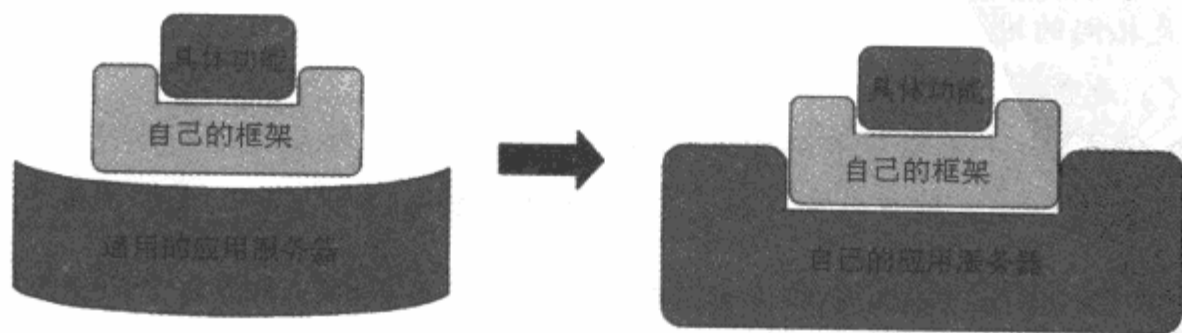


图 8-7 平台王道的过程

### 8.3.2 项目的分割

项目的分割是指在软件开发过程中对一个项目的各个模块功能进行合理的分割，使每个模块独立开来，尽量减少模块间的耦合度，使每个部分各司其职，从而高效率保质量地完成工作。项目的分割不仅仅适用于大项目，对于小项目做分割也会提高开发效率。

“说完了平台，我们来谈谈项目的分割方式。”

“项目的分割是不是指把项目分工一下给大家做啊？”

“不不不，项目的分割远不止分工这么简单。项目的不同分割方式对于项目的开发起到的促进或滞后作用也不尽相同。软件开发的历史中出现了许多种项目分割方式，我们今天就来回味一下过去，总结一下现在，展望一下将来，呵呵。”

“嗯，咱们也来一个‘昨天、今天、明天’！”

软件开发中项目分割的方式，大致可以分为如下三种。

- 大水平，小垂直
- 大垂直，小水平
- 面向方面的分割

#### 1. 大水平，小垂直

以一个公司的管理系统为例，“大水平，小垂直”的分割方式如图 8-8 所示。需要注意的是这种分割方式中是以水平分割为主，在水平分割后产生的各个子系统中需要对项目进行再一次的垂直分割。水平项目分割为主的思想是优先按照软件系统的功能模块进行划分的。

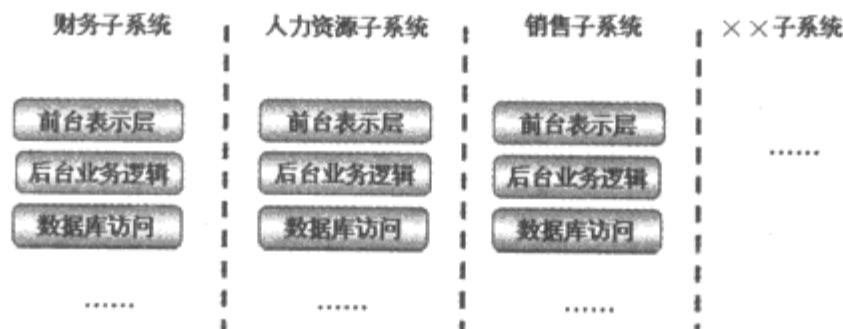


图 8-8 水平项目分割

“师兄，我觉得水平分割为主还是挺合理的呀，它有什么缺陷吗？”

“水平分割为主的缺陷是会造成大量的重复开发，比如财务子系统需要有人做数据库访问、后台业务逻辑、前台表示等工作，而到了人力资源这些工作还是需要去做的，这些东西虽然属于不同的部门，但是相同的地方很多，这样开发就很烦琐了。”

“哦，说得也是啊。”

“而且你想想看，每个人所擅长的肯定都不一样，这样按照功能先进行水平分割，就要求每个子系统的项目开发小组在数据库访问、后台业务逻辑等方面都要有比较平均的高水平。”

水平分割为主不适合做特别大的系统，而且水平分割也使得一些某个领域比较精通的人无法发挥自己的优势。比如一个开发人员表示层方面的技术如 AJAX 等非常擅长，但是由于项目是水平分割的，这个高手在完成了自己在表示层的工作后，还必须去开发自己并不擅长的数据库模块。

水平分割是一种比较古老的分割方式，这种分割方式现在已经使用得不多，因为水平分割没有将共性的东西提取出来，不利于开发人员专注于某个领域发挥自己的特长。

## 2. 大垂直，小水平

“大垂直，小水平”的项目分割如图 8-9 所示，和水平分割为主不同，垂直分割为主是先进行模块的垂直划分。如将所有的子系统业务逻辑集中到一个层面上来开发，再在这个层面将其分为财务方面的业务逻辑、人力资源方面的业务逻辑等。

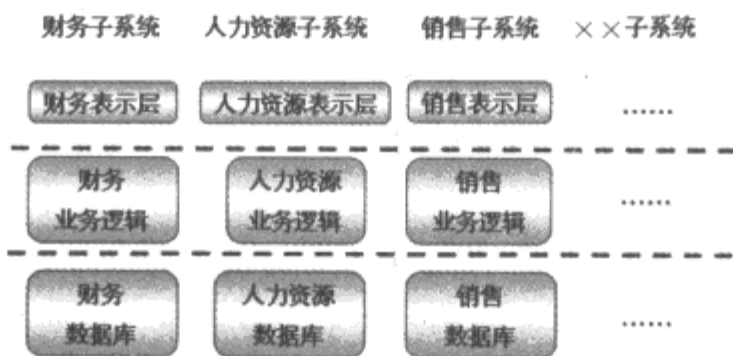


图 8-9 垂直项目分割

“师兄，既然垂直分割为主比水平分割为主要先进，那它的优势是什么呢？”

“垂直分割的思想和平台的思想类似，它做到了把开发项目中共性的部分单独提取出来，如不管是财务部门还是人力资源部门，他们对数据库方面的安全性、高可用性、数据库事务等方面的要求是一样的，所以没必要针对每一个部门的子系统都开发一遍。”

“嗯，这么说来垂直分割的确好处多多啊。”

“是啊，项目垂直分割之后某一个大家共同的领域就会有人专门去做，这样有利于发挥每个人的长处，比如负责数据库开发的人，不需要关心其他人，而他自己也不需要对其他领域的知识有很深的了解，只关心自己的数据库方面就行。”

垂直分割项目是按照计算机技术的领域不同进行划分的。这是一种当下流行的分法。如果项目组中的每个人的强项能力指数为 8 的话，那么所开发出来的项目则至少是 8 级。垂直分割方式可以很好地采用当下流行的一些平台组合，比如表示层用 JSF，业务层用 Spring，数据库持久层用 Hibernate。

## 3. 面向方面的分割

面向方面的项目分割如图 8-10 所示；它是由垂直分割进一步发展后得来的。在对项目进行先垂直后水平的分割之后，如把业务层分为财务业务逻辑、人力资源业务逻辑等业务逻辑之后，发现就算是分割后的小模块，还是有很多共性的地方存在。



图 8-10 面向方面的项目分割

这些共性的地方如系统日志、安全性、数据库事务并发等方面，使得项目的重复开发负担还是不轻。而且不仅仅是业务层需要这些功能，数据持久层和表示层也对如系统日志、安全性等系统级的功能有很大的需要。而这些功能对每个垂直层而言本质上是一样的，重复开发很不合算。

“蔡佳娃，现在向你介绍一个新的编程思想——面向方面的编程（Aspect-Oriented Programming, AOP），这个是在面向对象编程思想之后提出来的。”

“啊，又是一个新概念啊，师兄赶紧给我讲讲吧。”

“面向对象的编程你应该很熟悉了吧，面向对象编程以对象的特性来考虑问题，也就是由不同的对象负责不同的工作，只要处理好不同对象之间的消息传递和功能调用即可。但面向对象的编程方式对系统横切关注点的处理效果并不太理想。”

“什么是系统的横切关注点呢？”

“系统的横切关注点区别于业务逻辑，项目中不同的模块有不同的业务逻辑，而横切关注点则星罗棋布般地渗透在每个模块当中。在系统的整个业务逻辑中，到处都将涉及横切关注点，如系统日志、安全性等。”

如果按照传统的面向对象编程模式，则在每个功能的实现类中，都要重复地编写系统横切关注点的代码，这使得开发过程烦琐累赘，同时也降低了代码的可维护性。

由于横切关注点关注的是系统级的服务，这些服务对于大部分应用而言是基本相同的，所以可以采用 AOP 技术来实现，例如将系统中安全性、数据库事务、系统日志等公共服务提取出来封装到一起，这样就提高了代码的重用性，减少了各个模块的耦合度，使系统结构更加直观。

“面向对象编程是将项目按照层次结构分解成不同的对象，而 AOP 技术是将系统分解成不同的方面来看待，使得开发人员可以从不同的角度来看待问题。”

“哦，那 AOP 技术在真实项目中实现需要什么框架的支持呢？”

“目前业界还是有很多框架实现了 AOP 技术的，如 AOPSpring、AOPAspectJ、Aspectwerkz 等。”

## 8.4 “大而全”还是“精而深”

IT 职场中的技术人员对于知识的掌握情况，大概可以分为“大而全”和“精而深”两个方面。每个开发人员进行技术知识的储备时，或许都会在“大而全”和“精而深”两者之间取舍徘徊，但究竟“大而全”和“精而深”是一个开发人员成长过程的不同阶段，还是最终达到的不同境界呢？本节就来讨论一下这个问题。

### 8.4.1 “大而全”和“精而深”矛盾吗

很多人把“大而全”和“精而深”看做是孟子所言的“鱼”和“熊掌”，认为虽然皆我所欲也，但是“二者不可得兼”，选择了一个就没有办法兼顾另外一个。这种想法错误地把“大而全”和“精而深”放在了矛盾的对立面上。

“师兄，你说我是做个‘大而全’的开发人员好呢？还是要‘精而深’？”

“嗯？此话怎讲？”



“现在不都讲究技术储备、终身学习嘛，我得给自己找对方向啊！你说呢，师兄？”

“我说你大概没弄明白‘大而全’和‘精而深’到底是怎么回事吧？”

“哦？不是两种人才的发展方向吗？”

“还是让我来给你具体讲讲这两个方面吧。”

很多人认为“大而全”和“精而深”是对立的，其原因在于这些人是用错误的方式来进行“大而全”或“精而深”的技术钻研，主要表现在以下两个方面。

- 不着边际的“大而全”
- 没有基础的“精而深”

### 1. 不着边际的“大而全”

“师兄，什么是不着边际的‘大而全’啊？”

“不着边际的‘大而全’是指在提高自己的时候对于一些技术的研究过于广泛，比如今天研究Java，明天钻研C++，后天征战C#，这样的确是做到‘大而全’了，但是把战线拉得过长，也会使自己显得分身乏术。”

“是啊，研究太多了肯定是吃不消的。”

“还有一种不着边际的‘大而全’就是对于一些基础或低层次的技术进行重复钻研，比如今天研究椭圆按钮的实现，明天又去发明其他视觉样式的控件，以这种方式修炼，时间是充分付出了，但是收到的回报却总是只有那么一点。”

不管是打一枪换一个地方的漫天“大而全”，还是重复、过分地投入于基础知识的“大而全”，都是不正确的“大而全”。这种学习方式的结果就像是漫无边际的打地基，地基打了一大片，却没有上层建筑，这种工程显然没有实用价值，如图8-11所示。

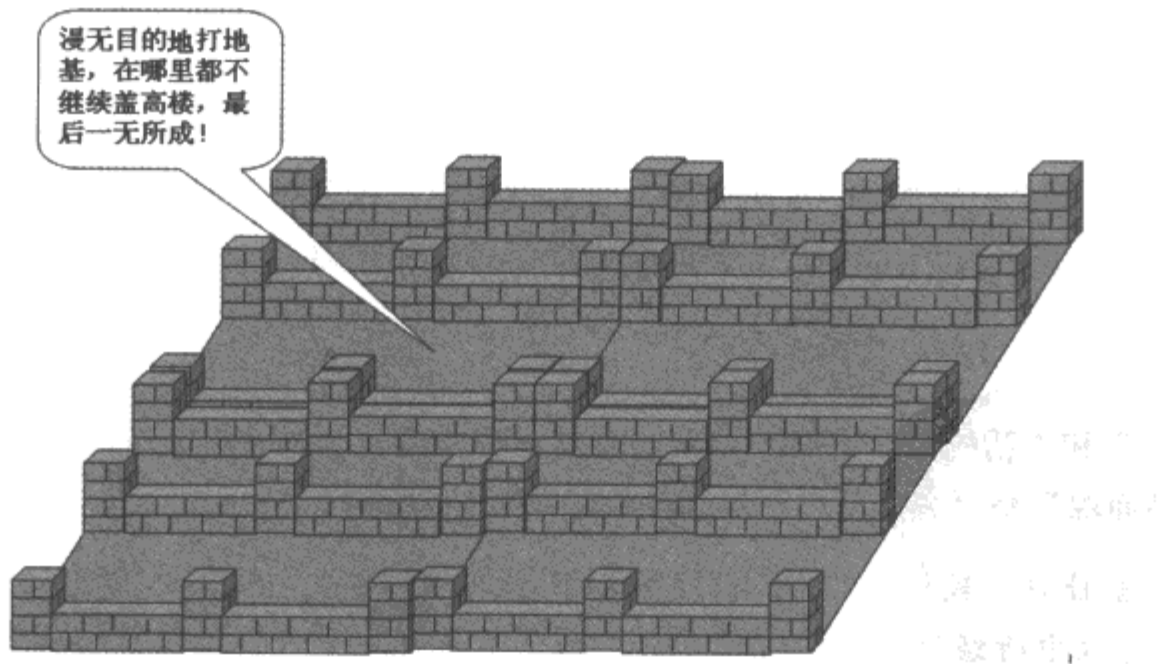


图 8-11 不着边际的“大而全”

在这种费事不讨好的盲目“大而全”之后，当然就没有精力再进行“精而深”的研究了。这样就难免产生“大而全”和“精而深”只能二者选其一的感觉了。

在学习中对有价值的内容进行“大而全”的研究，对意义不大的内容进行太广泛的研究也没用。比如身为Java开发人员，在精通Java SE的基础上，掌握手机端的开发和企业级的开发，

这就是比较有价值的“大而全”。而对于其他方面，就应当避免花费过多的精力。正像本书在前面提到过的，学海无涯，但人生苦短，应当把有限的精力用在最有意义的地方。

## 2. 没有基础的“精而深”

“说了不着边际的‘大而全’，再来看看没有基础的‘精而深’。”

“哦，那是什么样的情况啊？”

“与不着边际的‘大而全’相反，这种‘精而深’对于基础并没有过多的重视，只是一味地对于某个技术门类进行死磕。”

“钻研，当然要深一些嘛。”

“需要明白的是，我们所说的‘精而深’不是在地上挖一口井，而是在地上盖一座高楼，挖井不需要基础，只管狠命往下挖就行。但是盖高楼如果没有基础或是基础不够扎实可就很危险了，很可能经不起强震或飓风而颤颤巍巍地倒下。”（如图 8-12 所示）

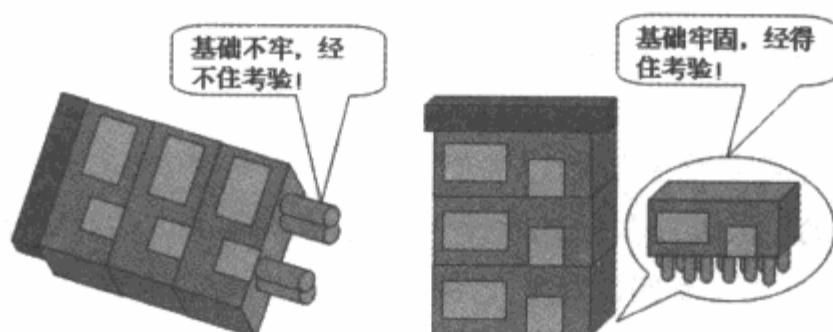


图 8-12 没有基础的“精而深”

这种没有基础的“精而深”也是让“大而全”和“精而深”陷入矛盾的幕后黑手之一。例如，很多人在学习某项技术的时候，对其中的某一个方面进行深入研究，发现其深奥无比，下了狠工夫都不见得能够大彻大悟。于是便仰天长叹曰：如此精深的技术吾绞尽脑汁尚不能解其一面，如何再有精力和时间去触类旁通，达到“大而全”的境界呢？

很多初学者在学习的时候就会产生这样的想法，因为他们喜欢把学习中遇到的感兴趣的部分深究一下，但是很多时候都是以失败告终。原因就是自己的基础并没有达到能够接受深层次知识的程度，这也就说明了“大而全”其实就是“精而深”的基础。

### 8.4.2 “大而全”托出“精而深”

“大而全”和“精而深”不仅不矛盾，而且从前文也可以看出，“大而全”是“精而深”的前提条件，“精而深”是“大而全”达到一定阶段的必然跨越。

“好了，蔡佳娃，我们来总结一下，像我们前面所说的，‘大而全’和‘精而深’是一名开发人员成长过程中所经历的不同阶段。它们两个不是互斥事件，是相辅相成的，是可以都做到的。”

“嗯，应该是从‘大而全’逐步过渡到‘精而深’对吧？”

“是的，‘大而全’到一定程度以后，应该对自己已经掌握的知识做一个审查，选择自己最擅长、最感兴趣的方向深入钻研，这样一步一步地走下去，就会达到‘精而深’的境界了。”

“这应该是许多高人的成功之路吧。”

“是啊，很多 IT 历史上的英雄能够选择自己的研究方向并在相关技术领域获得骄人的成绩之前，都首先是一个‘大而全’的人才。”

“大而全”和“精而深”的关系如图 8-13 所示，没有金字塔底“大而全”的支撑，就没有塔顶“精而深”的光彩。在进行了有意义的“大而全”的学习之后，有选择地从其中挑出一部分进行精炼，才能更有效地进行技术的修炼和提升。

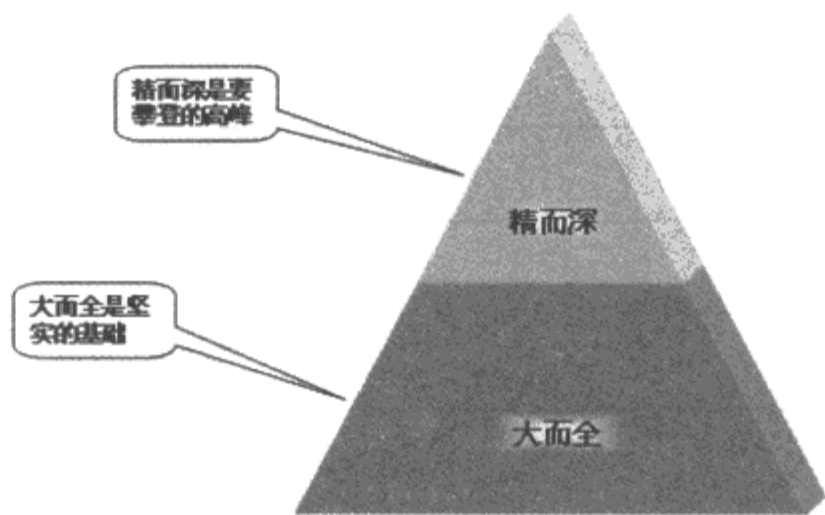


图 8-13 “精而深”是“大而全”的基础

计算机行业鲜有只精于一道的专家，通晓诸子百家学说的全才也很罕见，真正的高人应该是对于和自己相关的领域该知道的都知道，并在此基础上精通至少一两项。每个开发人员都应该有攀登金字塔的勇气和耐心，先打好基础，然后在坚实的基础上不断地向新高度进发。

### 8.5 本章小结

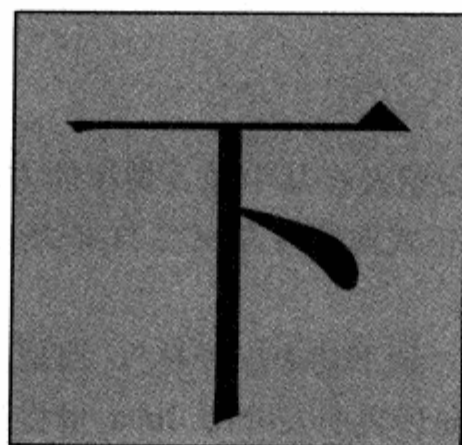
江湖多歧路，有的路越走越平坦宽敞，有的路则会越走越崎岖狭窄。有的路难走也就罢了，可能还能通向康庄大道，有的歧路搞不好就会把人带到悬崖边上。走的路线不同，对于一名开发人员的发展所造成的影响也不同。因此，行走江湖，还是要大胆行事，谨慎择路。



## 学习笔记栏



P A R T



---

## 下篇 笑傲江湖

---

第 9 章 天下功夫出少林

第 10 章 几种自废武功的做法

第 11 章 没有必杀技，怎么敢出来混

第 12 章 新锐兵器谱

第 13 章 武学奥义

第 14 章 杂项



# 第 9 章 天下功夫出少林

本书前面几章讲述了 IT 职场的现状及在其中做人做事的方法与态度，提出了一些有关如何修炼自己的建议。从本章开始，本书将正式向读者介绍在 IT 职场中生存发展所需掌握与关注的 Java 技术及能力方面的知识。

武侠小说中经常讲这样一句话：“天下功夫出少林”，那是因为少林功夫最正宗、最实用。本章就姑且作为整本书的“藏经阁”，向读者朋友介绍 Java 世界里最正宗、最实用的一些技术，身为一名 Java 开发人员，这些少林功夫是必须要掌握的。

## 9.1 Java EE 开发人员必知必会

Java EE 的开发人员，算是 Java 开发者大军中为数最多的一类人了。从最普通的程序员，到 ERP 架构师，具备什么样的素质才能够让自己驰骋于这片广袤的大地呢。本节就来谈一谈身为 Java EE 开发人员所必知必会的知识，希望对读者朋友有所帮助。

### 9.1.1 坚实的基础——核心 Java

在 Java 技术中，核心 Java 是最容易被初学者忽视的内容。其实一些 Java EE 开发人员在工作中阻力重重，力不从心，很多时候并不是框架或其他技术没有掌握好，而是自己前期的核心 Java 基础没有打好，导致后期的工作后劲不足。

“蔡佳娃，从今天起，我们要开始谈谈 Java 技术中的真功夫了。”

“好啊，终于可以得到师兄的真传了。”

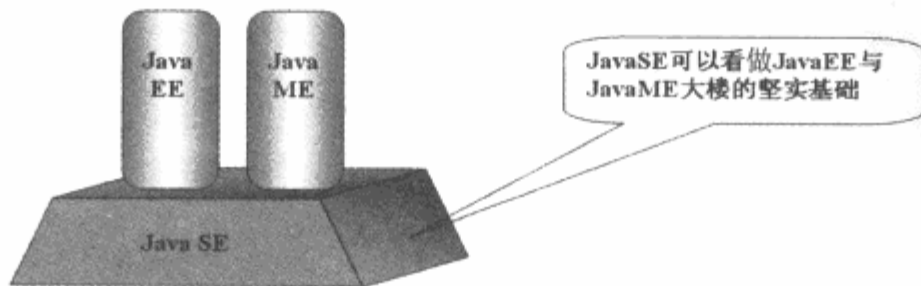
“Java 的开发人员主要分为两大类，Java EE 开发和 Java ME 开发。考虑到你目前的职业，我们先来谈谈 Java EE 开发人员的必备技能。”


“好啊，正好我来对自己做个检查，看看哪里还不够强悍。”

“OK，首先要谈的一个技能就是核心 Java 的掌握。”

“咦，为什么核心 Java 会是第一位呢？”

不管是从事 Java EE 开发，还是 Java ME 开发，核心 Java 都是必须要首先掌握的，三者的关系如图 9-1 所示。如同盖高楼最重要、花费也最高的是打地基一样，学 Java 就应该把基础打实。如果对核心 Java 重视得不够，研究得不彻底，还想要成为 Java 大牛，那就有些异想天开了。



 **提示** 这里有必要区分一下 Java SE 和核心 Java 的区别,Java SE 中抛去界面编程如 Swing、Awt 等技术剩下的内容基本属于核心 Java,其实核心 Java 可以看做是 Java SE 的子集。

“蔡佳娃,你竟然敢小看核心 Java,那我就让你见识一下藐视核心 Java 的后果。”

“啊,不会吧,师兄很生气,看来后果很严重啊。”

“给你一个数组,用 Java 语言开发排序程序,你怎么弄?”

“就是用排序算法呗,快速排序之类的。”

“看来你果然对核心 Java 的类库掌握不周全,你难道不知道 Java 的 util 包下面有个 Arrays 类吗?其中就有一个 sort 方法,可以对指定的数组进行排序,比你自己写要强多了吧。”

“哦,这个还真不知道。不过我有个问题,这个排序函数是按照什么顺序进行的呢?升序还是降序呢?如果我需要排序的数组不是基本数据类型而是自定义的对象引用呢?”

“你能提出问题我很欣慰,不过如果你对类库掌握得好,就不会问这些问题喽。”

Arrays 的 sort 方法已经被重载了,既可以对基本数据类型数组进行排序,同时也可以对对象数组进行排序,这时对象就必须即实现 Comparable 接口,使对象自身有自然顺序。以上两种排序都是按照升序的规则进行的,如果要用自定义排序规则就需要为 sort 方法指定一个比较器(Comparator)。

除了排序方法,Arrays 类中还提供了其他很实用的数组操作方法。如二元搜索、数组比较等,这些都为开发人员的工作带来了不少便利。

“蔡佳娃,大学时上数据结构肯定接触过特大整数的加减乘除运算吧,还记得是怎么做的吗?如果工作中需要解决此类问题怎么办?”

“嗯,我记得当时用的是数组或者链表来完成这些特大整数的运算的,不过这应该只是用来帮助理解和设计算法的练习题吧,实际开发中遇到这种情况的时候多吗?”

“怎么说呢,实际开发中遇到这种情况的时候还真是不多。另外还有一个更不容易处理的就是特大浮点数的运算。”

“啊?浮点数要比整数复杂得多呢。”

“出于种种原因吧,计算机在表示浮点数的时候可能会出现误差,而这些都是金融、财务等行业所不能忍受的。因此,在开发中也是可能遇到这样类型的问题的。”

“那就只好去做喽,用数组或者链表堆砌呗。”

关于特大整数、浮点数的运算,Java 中专门提供了 java.math.BigInteger 类和 java.math.BigDecimal 类。通过将特大整数或浮点数封装到这两个类中,开发人员可以方便地对其进行加减乘除取模等算术运算。这两个类对于数值的大小和精度是没有限制的,能很好地满足用户对数据范围和精度的要求。

另外读者朋友要注意,类库中的功能可是大量大牛辛勤工作的结晶,可靠性与速度都是很有保证的。如果不全面了解类库,而自己去开发诸如此类的功能,一方面会很困难,另外速度、可靠性也很难保证。

“蔡佳娃,我再来考考你,看看你对于核心 Java 掌握得如何。”

“师兄，你饶了我吧，我认输了，我不该小看核心 Java 的重要性啊。”

“呵呵，好吧，我给你讲讲我初学 Java 时的一段经历吧。”

“好啊，只要不再对我那点可怜的知识严刑拷问就行。”

“我当时草草地学完了 Java 之后，想自己做出个小游戏来练练手，最后决定做一个纸牌游戏，编写代码的过程中我遇到了一个问题，就是在第一轮游戏之后需要洗牌，我对于洗牌这个算法的研究可以说是辗转反侧了好久，但一直没有想出好的解决方案。”

“是啊，排序好说，打乱顺序有时还真不容易。”

“我将每一张牌设计成一个对象，54 张牌装在一个集合中。我后来忍不住了，去网上发了个帖子问问，结果一高人回了一句：知道 Collections 类的 shuffle 函数么？我回去一查，这个 shuffle 函数就是用来打乱集合中元素顺序的。”

“啊，看来核心 Java 类库中的好东西还真不少。”

除了故事中提到的那些，核心 Java 中有很多东西能够助开发人员一臂之力。例如 JDBC 中提供的可以获取数据库和结果集元数据的方法，可以让开发人员对数据库的结构信息有一个整体的把握，使开发更为灵活方便。再如早期为了提高线程利用率需要开发人员自己开发线程池，而现在 Java 类库中提供了各种各样的线程池，足以满足市面上开发的需要。

上述这些都是核心 Java API 当中的，是 Java 中进行各种基本操作的宝库。只有掌握了这座宝库，对其中的各个宝贝有深入的了解，才能自信地说：“我是一名合格的 Java 开发人员。”

“蔡佳娃，很多人都是由于对 Java 类库 API 不够了解，自己重复发明车轮，结果造成开发速度变慢，开发成本增加，代码质量也有所下降。因此在那次洗牌事件之后我痛定思痛，下定决心好好研究核心 Java，如果没有那件事，我不可能有现在的成绩。”

“是啊，我们的技术水平再牛，比起 Sun 的专家们来说不知道要差多少倍，怎么会开发出更高明的代码呢？看来我对核心 Java 类库的熟悉度也确实不够啊。如果没有师兄，我也不会明白自己一直以来所欠缺的地方呢。”

“我以前不也给你讲过我的一个师兄嘛，毕业时他就会核心 Java，不过不是一般地会。人家当年参加工作后就是凭着浑厚的基本功在两年之内达到年薪十几万的。”

多年的学校教育让许多同学产生了一种错觉，认为基础的都是简单的，其实并非如此。诺贝尔奖就经常颁发给对各个学科进行基础研究的科学家们，基础才是最应该投入精力和时间去学习的，核心 Java 不仅学起来不简单，而且还非常重要。

需要说明的是，核心 Java 可不仅仅包括 Java 类库的 API。除了这些，想掌握好核心 Java 还需要深入学习面向对象的编程思想，否则很难驾驭好 Java 这门功能强大的语言。

“蔡佳娃，精通核心 Java 除了要熟练掌握这些 API 之外，还需要学习面向对象的思想啊。”

“哦，这也属于核心 Java 吗？”


“当然属于了，身为开发人员，是不能把语言本身和贯穿于语言的思想分离开的，Java 的强大与否取决于我们如何使用它。如果你把它当做 C 语言来使唤，把数据和方法分割开来而不是封装，那掌握再多的 API 也只是在为结构化编程服务，而不是面向对象的开发。”

“的确，语言说白了还是工具，产生什么样的效果就看我们怎么用了。”

“说得很对，而且不像刚刚谈的类库 API，思想这个东西很难讲出来，也很难传授，所以你要多多体会和领悟，因为这些可能比 API 还重要。”

语言是思想的载体，中华民族的语言发展至今，就承载了浩瀚五千年的文明思想，不懂中华文明的思想是无法学习好汉语的。同理，不懂面向对象最基本的多态性就想着要去搞 Spring、搞 AOP，肯定也是不会有任何成果的。

因此，想要征服 Java 让其乖乖地为自己服务，就必须要对其承载的面向对象思想深深挖掘和细细体会了。在这里也可以向读者推荐一本介绍面向对象思想理论的大部头——《面向对象方法：原理与实践》，有兴趣的读者可以啃一啃，提升提升自己的理论水平。

 **提示** 还有许多核心 Java 的高级问题，本书将会在后继的章节中再为读者朋友们介绍一些。

### 9.1.2 只会 Java 可不行——大牛的百宝囊

要想成为一名合格的 Java EE 开发人员，只会 Java 可是不够的，再强大的语言也不能面面俱到。每个 Java 大牛的身上都会有一个百宝囊，里面装着成就高人的技术宝藏，本小节将为读者打开这个百宝囊一探究竟。

“师兄，要是把你刚才说的核心 Java 给掌握了，算到了什么境界啊？”

“用《大腕》里面的话来说，掌握了核心 Java，你只能说自己学过 Java 了，要不然你在路上都不好意思跟别人打招呼。”

“那要成为一名 Java EE 工程师，还需要什么知识呢？”

“以我的经历来谈的话，想要在 Java EE 界闯出一番名堂，在学习了核心 Java 的基础上还需要掌握的东西有 SQL、XML、HTML、CSS、AJAX 等技术，下面我们来一一介绍。”

#### 1. HTML

HTML(Hyper Text Markup Language, 超文本标记语言)读者应该都很熟悉,如果有人对 HTML 感到陌生的话那他就太需要恶补互联网的知识了。HTML 是在浏览器中解释渲染的,所以不管是微软的 Windows 系列,还是苹果的 Macintosh,有浏览器的地方就有 HTML 的身影。

作为最早和应用最广泛的网络编程语言,HTML 是每个进行 Java Web 开发的人都必须熟练运用的语言。HTML 通过提供各式标签,将网页的骨架搭好,剩下的工作则由 JSP、CSS、AJAX 等来完成,从而构建出一个提供强大功能的华丽 Web 界面。

相对于后面要介绍的其他技术,HTML 相对好学一些,而且使用起来也没有那么多的限制,需要注意的是 HTML 在浏览器中解释渲染的时候不会报错,而是会将错误略过,这在提供方便的同时也给开发带来了一定的不便。

HTML 标准由 W3C 组织制定,目前的最新正式版本是 4.01。XHTML 是基于 XML 的 HTML 改进版,旨在提高与 CSS 的结合度。尚未发布的 HTML 5 目前仍是草案,其将大大增强 Web 的表现能力。

#### 2. CSS

CSS(Cascading Style Sheets, 层叠样式表)是 Web 开发中另一个需要开发人员重视的利器。

通过 CSS，可以大幅度地提高 Web 页面布局和排版的效率。CSS 通过对文字的颜色、字体、图片的摆放等方面的灵活设置，将内容与显示方式剥离开来，大大增强了代码的可维护性。

CSS 不关心所要显示的内容，只负责网页内容的表现方式，通过制定不同的规则，将网页内容按照统一的样式进行排版。CSS 也是由浏览器解释渲染的，可以内嵌在 HTML 代码中或者单独放在后缀为 css 的文件中。

CSS 再加上 XML 和 JavaScript 等技术，就构成了目前 Web 开发中最火爆的应用技术——AJAX。所以如果对 AJAX 感兴趣，CSS 就不得不非常精通了。

### 3. AJAX

AJAX (Asynchronous JavaScript and XML，异步 JavaScript 和 XML) 是目前最为时尚的 Web 开发技术。作为成就了 Google Map、Flickr 等众多互联网公司的技术，AJAX 提供了一种全新的客户端与服务端的交互模式。很多时候，AJAX 就是 Web 2.0 的代名词。

AJAX 通过改善传统的 Web 请求响应模式，使得不需要全部刷新页面即可更新数据，大大提升了用户的 Web 体验，使得 Web 应用仿佛如桌面程序一般。同时 AJAX 的后台除了是 Java 平台，还可以是 PHP、.NET 等 Web 平台。

AJAX 不是一门新的技术，AJAX 将市面上的很多技术整合到一起，如 JavaScript、XML、CSS 等技术，从而构造出一个美妙的组合。但是由于各个浏览器对于 XML 和 CSS 的支持不同，也使得 AJAX 在开发的时候需要考虑的问题增多。

### 4. Struts、Spring、Hibernate、JSF 等 EE 框架

框架技术在本书中多次被提到，这是一个 Java 开发人员，尤其是 Java EE 开发人员必须掌握并且熟练应用的技术。在竞争激烈的 IT 职场，一个不知 Struts、Spring、Hibernate 等框架技术为何物的求职者将很难在 Java EE 的职场中生存下去。

使用 SSH 等框架，就像是站在巨人的肩膀上做开发，很多基础和公共的工作框架已经都完成了，而且整个开发模式也已经展现在开发者面前。如 Struts 框架就很好地实现了 MVC 设计模式，如果没有 Struts 框架，开发者在进行开发的时候就只能靠自己来编写那些代码了。

### 5. SQL

SQL (Structured Query Language) 是专门针对关系数据库的操作而开发出来的，SQL 也是一种程序设计语言，不过不像 Java、C/C++ 等编程语言面临着来自其他语言的竞争，SQL 是市面上唯一认可的关系数据库处理语言，也就是说这是每个开发人员的必修课。

身为一名 Java EE 开发人员，工作中所开发的任何商业项目没有不需要数据库后台支撑的。虽然 Java 的 JDBC 和其他技术如 JPA 在数据库操作方面提供了强大的接口和 API，但是对于 SQL 的能力要求可是丝毫没有降低。

要想成为 Java 技术界的大牛，SQL 必然是制约其能力上限的重要因素，实际情况是 SQL 不仅要会，而且功力还不能浅。绝对不可以停留在对于不足一百条数据进行的高效率和简单地增删改查这个初级阶段，必须有如联合搜索、嵌套检索等高级运用的能力。

### 6. XML

XML 是一门功能强大的语言，其虽然在名字上和 HTML 有些血缘关系，但是在能力上 XML



要远胜过 HTML，它可以应用的地方也比 HTML 要多得多。

在 Java EE 的开发中，XML 被广泛应用在服务器端，不论是 Tomcat，还是 GlassFish 或是 Weblogic，服务器端的配置文件都是 XML 文件，如果对 XML 不甚了解或是一知半解的话，那么想要精通这些应用服务器就很难了。

不仅如此，在后面将要介绍的 SOA 技术中，XML 也担负着不可或缺的角色，比如 SOAP 中的数据都是以 XML 格式传递的。

## 7. 正则式

一个软件系统在研发过程中，尽管有图片、声音等多种媒体类型的数据需要处理，但是和开发人员打交道最多的，还是文本字符串。而想要灵活熟练地处理越来越复杂的字符串，就需要对正则式有相当深刻的理解了。

正则式在对字符串进行分析和匹配处理时有着无可比拟的强大优势，例如在开发一个验证用户输入是否符合电子邮件地址格式功能的时候，如果直接写的话工作将繁重无比，而采用正则式处理则会带来意想不到的便利。

Java 提供了用于正则式处理的相关类和方法，用于进行字符串的模式匹配、查找替换、分析等操作。身为 Java EE 开发人员应该充分掌握这些类和方法，在实际工作中应用，避免别人几分钟搞定的事情要熬夜开发还有 bug。

## 8. JavaScript

作为当今互联网世界最为流行的 Web 客户端脚本语言，JavaScript 在网页的动态性、交互性等方面起着重要的作用。尤其由于其在如今风靡整个 Web 开发界的 AJAX 技术中的上佳表现，使得 JavaScript 成为最炙手可热的脚本语言。

JavaScript 和 Java 之间除了名字上相似之外，并没有太多的亲缘关系。不过作为一名 Java EE 开发人员，JavaScript 的脚本编写能力直接反映了其在 Web 客户端开发方面的功底，每个 Java EE 开发人员都应该像对待 Java 一样去学习 JavaScript。

很多 JavaScript 的初学者认为作为脚本语言，JavaScript 是比较容易掌握的。这就大错特错了，作为脚本语言，JavaScript 自然有其易学易用的方面，但是想要做一个 JavaScript 高手，就不能仅仅停留在一些网页特效上面。JavaScript 提供的很多高级特性如嵌入执行脚本、注射新方法属性、基于对象编程等都需要开发人员去掌握。



**提示** 部分在本小节简单介绍的知识在后续章节中有比较详细的介绍，本小节只是做一个概览，使读者有一个全局的把握。

## 9.2 Java ME 开发人员必知必会

前面一节介绍了身为 Java EE 开发人员必知必会的知识，本节将谈一谈从事 Java ME 的开发人员应该具备的技术和能力。说起来，Java ME 或许才是 Java 技术的“返璞归真”，因为 Java 最初就是被设计应用在嵌入式设备如电视机顶盒等家用电器上的。

需要说明的是，与 Java EE 开发人员相同，Java ME 开发人员也需要对核心 Java 这个基础有

着充分的掌握。不管到哪里基础都是第一位的，否则虽然面对的是规模比较小的移动程序开发，但仍然会因为基础的不牢靠而造成一些本可避免的错误。核心 Java 的重要性已经在前面章节中讨论过，本节将把重点放在合格的 Java ME 开发人员所应该满足的特定要求上面。

### 9.2.1 了解不同平台对 Java ME 的支持

Java 虽然以跨平台的优良特性而闻名，但与桌面程序和企业级应用不一样，Java ME 开发的手机软件虽然也是在运行在虚拟机（KVM）上面，但是由于各个手机制造商的虚拟机完全一样，而且具体到中国国情也会有山寨虚拟机的出现，因此相同的代码运行在不同手机平台上的效果也不尽相同。

考虑到这个方面，Java ME 的开发人员就必须了解不同的手机平台对于 Java ME 的支持差异。这样不仅避免了开发过程中遇到奇怪的现象而无法解释，而且还可以针对不同平台对 Java ME 的支持差异未雨绸缪，提前准备应对策略。

国庆假期，蔡佳娃抽个时间请师兄牛开复吃饭，一方面感谢师兄一直以来对他的照顾，另一方面也是和业内高人谈谈天，说说地。

“咦，蔡佳娃，你怎么一直低头看手机啊，忙着发短信呢？”

“嘿嘿，师兄你看，这个是我最近闲下来写的一个 Java ME 手机小游戏，呵呵，玩自己的游戏感觉就是不一样啊。”

“哦，是吗？用蓝牙传给我，我也瞧瞧。”

“咦，师兄，你的手机运行起来怎么跟我的效果不一样呢？”

“当然了，你的手机是联想的，我的是诺基亚的，平台不一样，效果当然也不一样了。”

“还有这一说吗？我还真不知道。”


“当然了。专业的 Java ME 的开发人员要了解不同平台对 Java ME 的支持程度。其实，Java ME 和 Java EE 一样，都是一种规范，在 Java EE 上可以有那么多的框架，在 Java ME 上面有一些不同的平台也就没必要大惊小怪了。”

不同平台对 Java ME 的支持差异其实也就是不同的手机厂商对于 Java ME 规范的不同实现，当今的手机市场格局在不断变化，就目前来看国内市场上的主流手机平台厂商有如下几个：

- 诺基亚
- 摩托罗拉
- 索尼爱立信
- 联发科（MTK）

诺基亚、摩托罗拉和索尼爱立信都在自身的软硬件平台上开发了基于 Java ME 规范的虚拟机，这些虚拟机最明显的不同就是对于控件在渲染方式上的差异。一般来说，上述公司的自主虚拟机对于 Java ME 中控件的诠释得要比 Sun 本身的 KVM 华丽得多。

联发科即 MTK 是大部分国内手机厂商和现如今铺天盖地的山寨手机厂商的解决方案，联发科的手机芯片可以支持 Java，而且要比国外其他品牌便宜许多。同时由于其提供了诸如双卡双待等符合中国市场的功能，使得联发科在中国的手机芯片市场一家独大。

 **提示** MTK 的手机解决方案是将手机硬件即芯片和手机的软件平台捆绑销售，这样手机厂商只需要对到手的 MTK 手机进行并不是十分复杂的二次开发，即可推出产品。这个特性也是其能够占领国内市场的一个重要因素。

下面针对几种不同的平台，从科技实力、开发过程、权威测试、技术支持几个方面做一个对比。

### 1. 科技实力

诺基亚、摩托罗拉和索爱等厂商历来对 Java ME 的规范支持得很全面，这大大提高了开发出的产品质量。同时有一些高端的手机可以支持 Java 3D 技术，有的甚至支持 Java FX 技术。

而联发科 MTK 的手机芯片很少能做到支持 Java 3D，大部分仍然停留在 2D 图形渲染能力的层次上。

### 2. 开发过程


索爱等国外的手机厂商都会针对自己的中高低端产品提供通用开发环境的模拟器插件或专用开发环境，这样可以先用专用模拟器实现，再到实际的手机设备上检验，在很大程度上保证了“所见即所得”的开发效果。有的手机厂商有自己的 SDK 用于提供一些专用的 API，如诺基亚。

而在国内市场占有率非常高、广大消费者都喜闻乐见的联发科平台却没有专用的模拟器，这样就很难保证所开发的项目是否能够在真实的手机设备上与通用模拟器保持一致。

### 3. 权威测试

国外的手机厂商由于具拥有自主研发的 Java 虚拟机，而且这些虚拟机也经过了权威机构对其是否符合 Java ME 标准的严格测试，因此在这些平台上开发应用程序只要遵从 Java ME 标准一般不会有任何问题。而采用了联发科解决方案的国产手机厂商并没有自己的虚拟机版本，一般也不是购买 Sun 的官方虚拟机，而采用的是 Sun 提供的一个虚拟机参考实现。

一个虚拟机的商用实现会考虑很多实际需要解决的问题，如性能优劣、美观程度、兼容性等。而 Sun 公司的虚拟机参考实现则在这些方面要差一些，而且外观样式不够华丽不说，性能也不够理想。所以在 MTK 平台上进行 Java ME 的开发就需要开发人员倍加小心了，必须先考查其虚拟机的版本情况以及性能问题。

 **提示** 联想公司是国内为数不多的采用品牌虚拟机的手机生产厂商。联想手机的虚拟机是从 JBlend 公司购买的，虽然在视觉样式上不能像索爱等公司那样为用户提供更高的体验，性能也可能相对差一些。但是对于 Java ME 规范实现的正确性还是能够保证的，从笔者的开发经验来看，JBlend 的虚拟机实现的外观比较接近于 Sun 公司的参考实现虚拟机。

### 4. 技术支持

基于联发科平台的手机厂商除联想之外对于 Java ME 的技术支持都不够，只对外声称支持 Java，解释得也比较模糊，甚至有的厂商根本就不提供技术支持。

国外的像诺基亚、摩托罗拉等厂商对自己的产品很负责，会将其支持的标准和不支持的标准明确详细地列出。而且这些大公司非常注重开发者社区的打造，为开发者提供了很多有意义工具、

服务和技术支持。同时，像诺基亚、索爱等公司还会召开技术大会，让人们了解技术的最新动态，真正地把开发者的热情点燃起来。

一个平台对 Java ME 技术的支持程度一般着眼于以下三点。

- 性能，即软件在其平台上运行效率。
- 美观，给用户的视觉体验。
- 兼容性，对于 Java ME 标准的支持程度。

所以在进行 Java ME 项目开发的时候，首先要搞清楚软件将要在哪个平台上运行，然后再考查该平台的虚拟机，这是比较重要的一步。如果有选择的余地，尽量选择一个有稳定虚拟机的平台来开发自己的产品。

## 9.2.2 游戏开发的基础知识

Java ME 技术的实现大多面向手机，而手机作为消费类电子产品，虽然会被用来工作，但是更多的场合手机还是用来娱乐的。而面向手机的娱乐方式，除了游戏，大概没有第二个产业能出其右了。从一开始的贪食蛇，到现在气势恢宏的手机 3D 游戏，在手机上娱乐的重头戏一直就是游戏。

既然如此，想要成为一名优秀的 Java ME 开发人员，游戏开发的知识就必须要掌握了。学过 Java ME 的读者都会知道，Java ME 最大的特点就是为开发者提供游戏编程的专用 API，这在 Java SE 与 Java EE 中都是没有的。由此也可以看出 Java ME 对游戏开发的重视程度。

“蔡佳娃，既然你对手机游戏还挺感兴趣，那我们就详细谈谈游戏开发的基本知识吧。”

“好啊，看来师兄你对 Java ME 这块也很精通嘛。”

“呵呵，哪有啊，只是做惯了 Java EE 的开发，闲下来的时候就想自己写几个小游戏玩玩，时间长了，也就积累了一些算不上宝贵的经验。今天咱俩就来分享一下做游戏开发需要的知识。”

“好啊，首先，Java ME 的开发人员也必须要掌握 Java SE 吧？这应该是基础嘛。”

“恭喜你，总算学会抢答了，呵呵。核心 Java 是两种开发人员行走江湖的必备基本功，不过这个问题我们今天就不再谈了，主要谈谈和游戏有关的基本知识。”

从技术实现上来讲，现在市面上存在的游戏主要三种，即 2D、2.5D 和 3D 游戏。本节将主要对使用 Java ME 技术开发手机 2D 游戏、2.5D 游戏的基本知识做一个简单的介绍，这些知识都是 Java ME 开发人员必知必会的。

### 1. 2D 游戏

2D 游戏的视角是从俯视的角度来看游戏场景的，如图 9-2 所示，一切立体的事物在游戏界面中都会转化成平面图形。

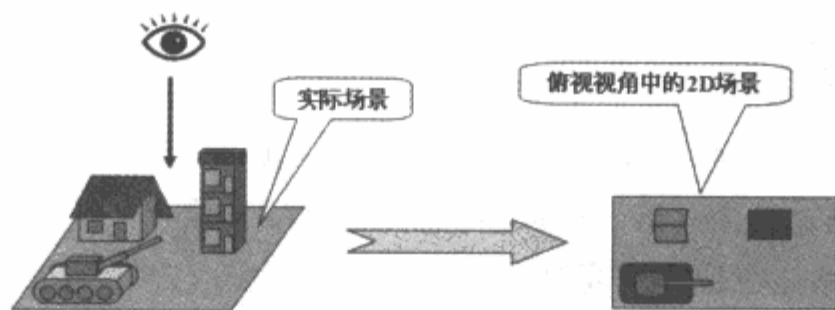


图 9-2 2D 游戏视角

2D 游戏是最早的游戏模式，使用 Java ME 开发 2D 游戏的技术已经非常成熟，其中专门为 2D 游戏的设计提供了专业的 API。开发人员使用这些 API 可以快速构建 2D 游戏场景，下面简单介绍几个游戏中常用的支持类。

• GameCanvas 类

GameCanvas 类是 Canvas 类的子类，为游戏提供了基本的屏幕输出功能。除了从 Canvas 继承下来的方法外，这个类还提供了游戏专用的功能。如查询当前游戏键状态的能力、同步图像输出等，这些功能大大简化了游戏的开发并提高了游戏的性能。

• Sprite 类

Sprite 类派生自 Layer 类，用来表示游戏中的精灵或怪物。Layer 类是个抽象类，代表游戏中的可视化元素。Sprite 对象可以显示一帧或多帧图像，这些大小相同的帧由一个 Image 对象在创建 Sprite 对象的时候提供，其原理如图 9-3 所示。

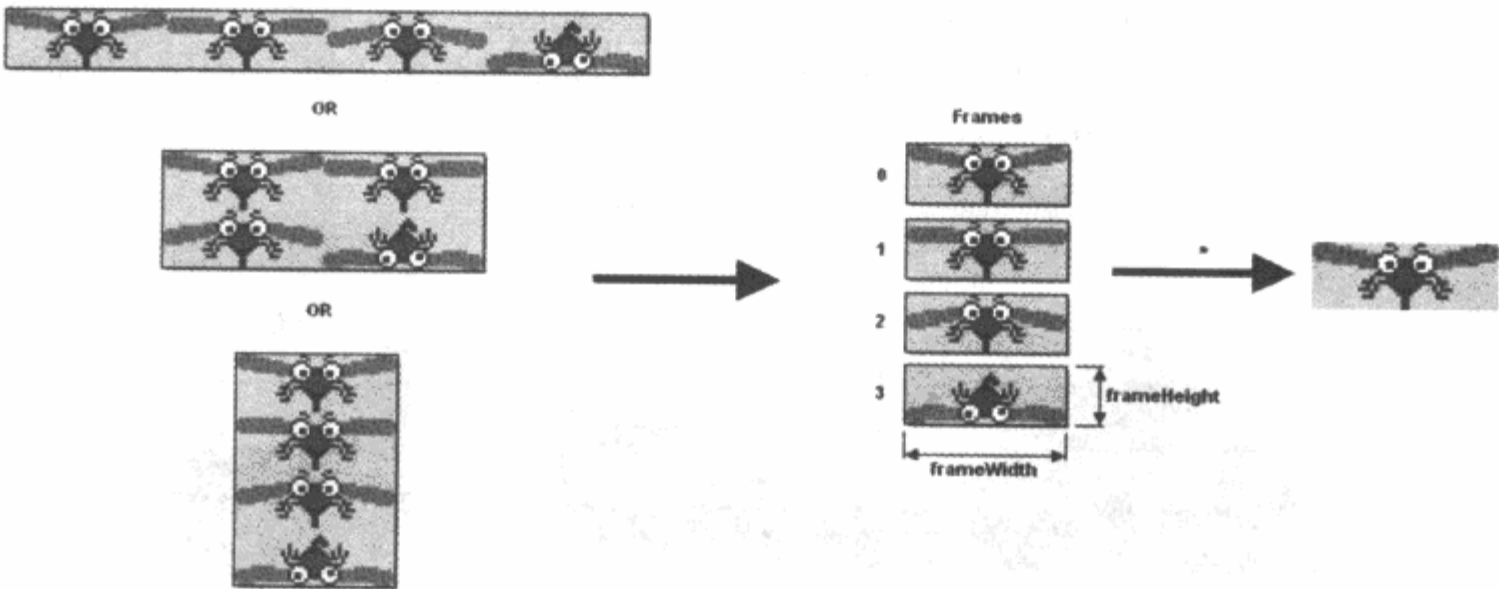


图 9-3 通过 Image 构造 Sprite 对象

从图 9-3 中可以看出，提供帧的图片中帧的排列顺序可以是任何形式，Java ME 会根据帧的宽度和高度按照从左向右从上到下的顺序自动进行切割。通过循环显示帧和改变帧的顺序，Sprite 对象可以实现复杂的动画，比如玩家行走时的动画或站立时斗篷的飘动等。

Sprite 类还提供了对帧进行翻转和不同角度的旋转等变换，同时通过设置 Sprite 的参考点可以更加方便地对 Sprite 进行定位。这些技术再加上碰撞检测方法，使得开发过程变得轻松简单，大大简化了游戏逻辑的实现。

• TiledLayer 类

TiledLayer 类允许开发者在不使用非常大的 Image 对象的情况下创建一个大的图像内容，如地图或背景图。一个 TiledLayer 对象由指定个数和大小的单元格组成，每个单元格称为一个图元。与 Sprite 对象类似，图元也是从一个 Image 对象中通过切割获取的。

TiledLayer 可以通过组合有限的图元表现出不同的视觉效果，如图 9-4 所示。例如游戏开发人员不必为游戏的每个场景单独绘制一幅地图，只需要将添加到 TiledLayer 中的不同图元进行排列组合即可。同时 TiledLayer 在构建大型动态场景的时候也非常方便，因为 TiledLayer 可以同时多个图元进行操作。



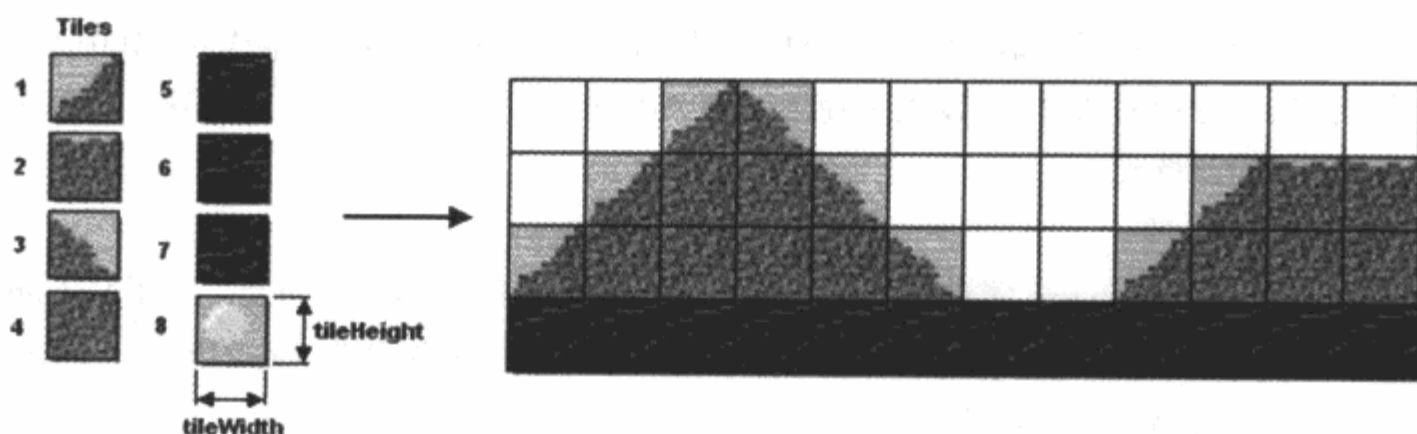


图 9-4 TiledLayer 图元组合示例

### • LayerManager 类

不管是 Sprite 还是 TiledLayer 都属于 Layer 的子类，而 LayerManager 类则负责实现分层次的自动渲染，从而简化游戏的开发。LayerManager 允许开发者设置一个可视窗口（View Window），表示用户在游戏中可见的窗口。LayerManager 的主要工作是自动渲染游戏中的图层，从而实现期望的视图效果，其原理如图 9-5 所示。

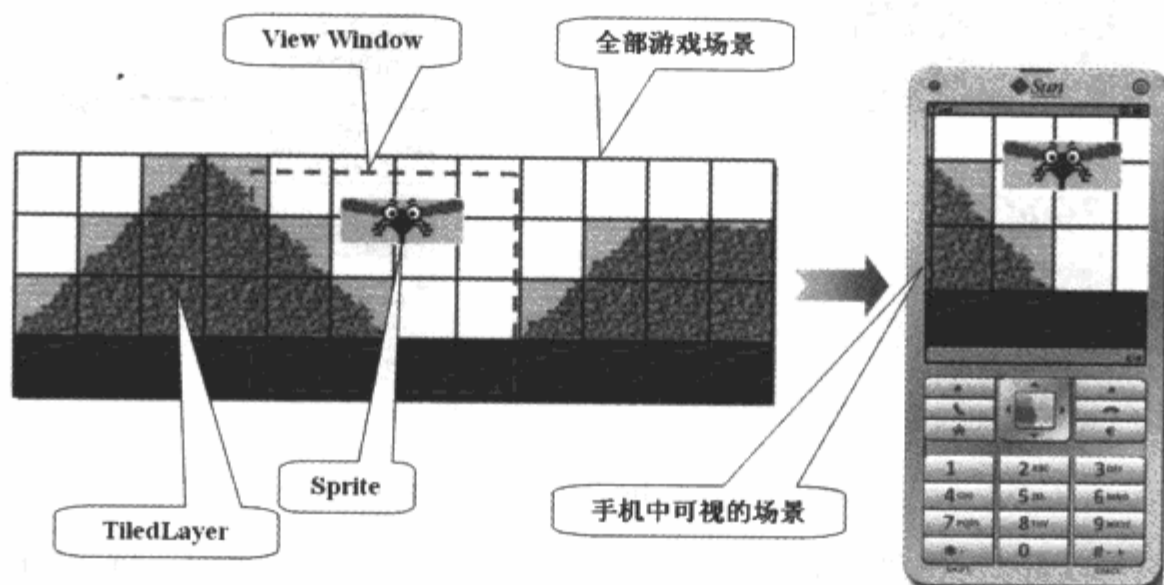


图 9-5 LayerManager 的工作原理

## 2. 2.5D 游戏

2.5D 游戏也是一种曾经风靡很长时间的 game 模式，之所以称其为 2.5D，是因为其视觉体验和技术水平都介于 2D 游戏和 3D 游戏之间。2.5D 游戏的本质是借助 2D 的技术实现类似 3D 游戏的效果，2.5D 游戏的观察视角如图 9-6 所示。

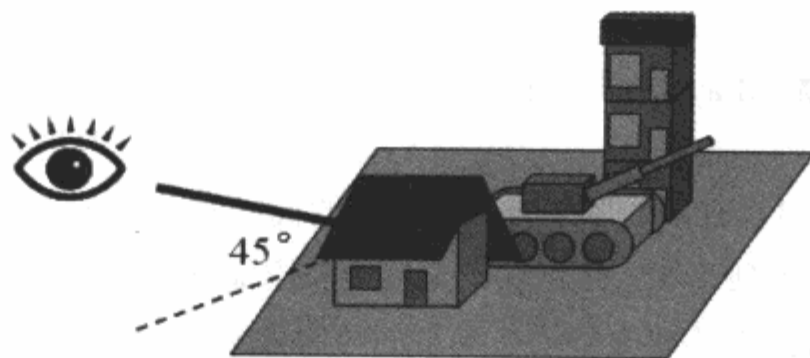


图 9-6 2.5D 游戏的观察视角

相对于 2D 游戏，2.5D 游戏最大的特色就是可以实现初步的 3D 立体效果。如图 9-6 中坦克会被平房遮住，楼房会被坦克遮住。这些视觉效果比 2D 游戏已经提高很多了，但是这个酷似 3D 效果的游戏界面却不可以改变视角，这也是 2.5D 游戏的最大缺点。

一个 2.5D 游戏的示例场景如图 9-7 所示，当然在真实游戏状态下只会看到整个场景的一部分。为了便于读者了解 2.5D 游戏是如何基于 2D 的技术实现 3D 游戏效果的，下面我们通过一个小例子来讲解其中的思想。

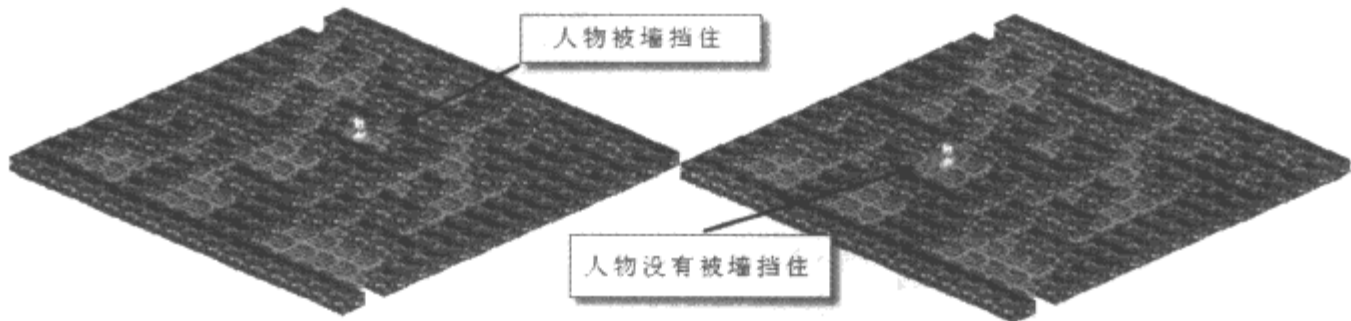


图 9-7 2.5D 游戏实现示例

2D 游戏的场景铺设是在每个单元格中填入不同的图元，2.5D 游戏也采用这种理念，不过与传统的 2D 游戏不同的是，2.5D 游戏中每个图元虽然在逻辑上只填充一个单元格，但在视觉效果上却不尽然。用平面图形表现层次立体感，这是 2.5D 游戏场景搭建的基本思想。

下面来介绍每个单元格的贴图规则，在 2.5D 游戏设计中，并不是所有的图元都会全区域填充像素，为了达到立体效果，图元中都会留出一些透明的地方，如图 9-8 所示为一个表现草坪的图元，其四周的透明区域在渲染图层时将不会给相邻的图元带来遮挡。

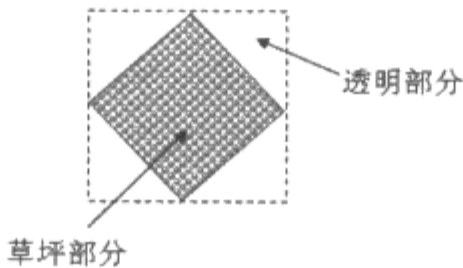


图 9-8 2.5D 图元示例图

假设有一个 3 行 3 列的场景，有 3 个图元需要按照设定的序号填充到单元格内，如图 9-9 所示。逻辑上每个单元格里面只填充一个图元。但是 2.5D 游戏中为了突出立体效果，图元的大小并不一定恰好为单元格的大小，图 9-9 中 1 号图元的大小（不包括透明区域）和单元格的大小相同，而分别表示建筑物和人物的 2 号和 3 号图元的大小则超过了单元格的大小。

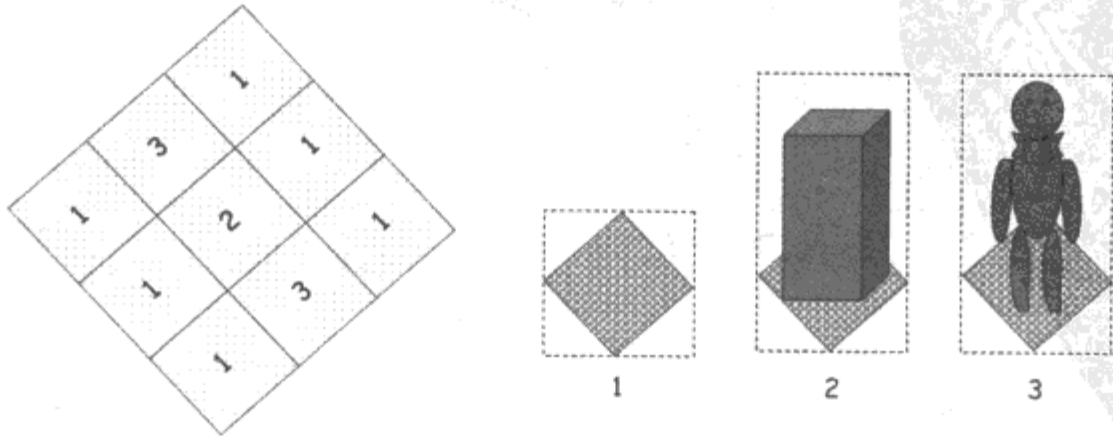


图 9-9 2.5D 游戏场景模拟图

可以将图元分为有效部分和超出部分，图元的有效部分决定其在场景中的哪一个单元格就位，而超出部分则会作为遮挡或被遮挡的部分去实现立体效果。

这就好比是棋盘，棋盘中的每个格子就是需要去填充的单元格，图元就是需要填充的棋子。棋盘的格子总是那么大，但是每个棋子的高度和大小变了，从棋盘的斜上方来看，就会有些棋子被遮住或者遮住别的棋子。2.5D 游戏场景的遮挡效果如图 9-10 所示。

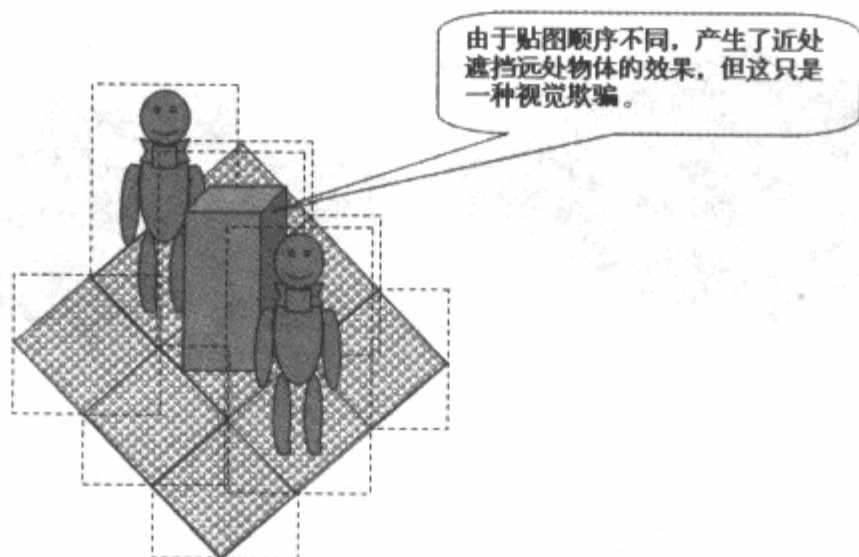


图 9-10 2.5D 游戏立体遮挡原理图

由于观察视角固定，所以 2.5D 游戏中贴图的顺序也必须是固定的，即从观察视角方向看，由远及近地将每个单元格内填充进图元，这样才能保证近处的元素挡住远处的元素。

### 3. 3D 游戏

2.5D 游戏虽然能够让用户体验到 3D 的立体感，但是由于 2.5D 游戏依赖于 2D 技术，而后者说白了就是在平面上的设计，同时由于 2.5D 游戏中每个单元格内的图元有着固定的遮盖顺序，所以无法切换视角，而能否切换视角才是 2.5D 游戏和 3D 游戏的本质区别。

3D 游戏可以通过编程真正地切换视角，其效果如图 9-11 所示。很多手机都支持 Java ME 的 3D 游戏标准，但都是些中高端手机，未来可能会有更多的手机支持 3D 游戏。判断手机是否能够运行 Java ME 平台下的 3D 游戏，要看其是否支持 Java ME 的一个标准——JSR-184，Mobile 3D。

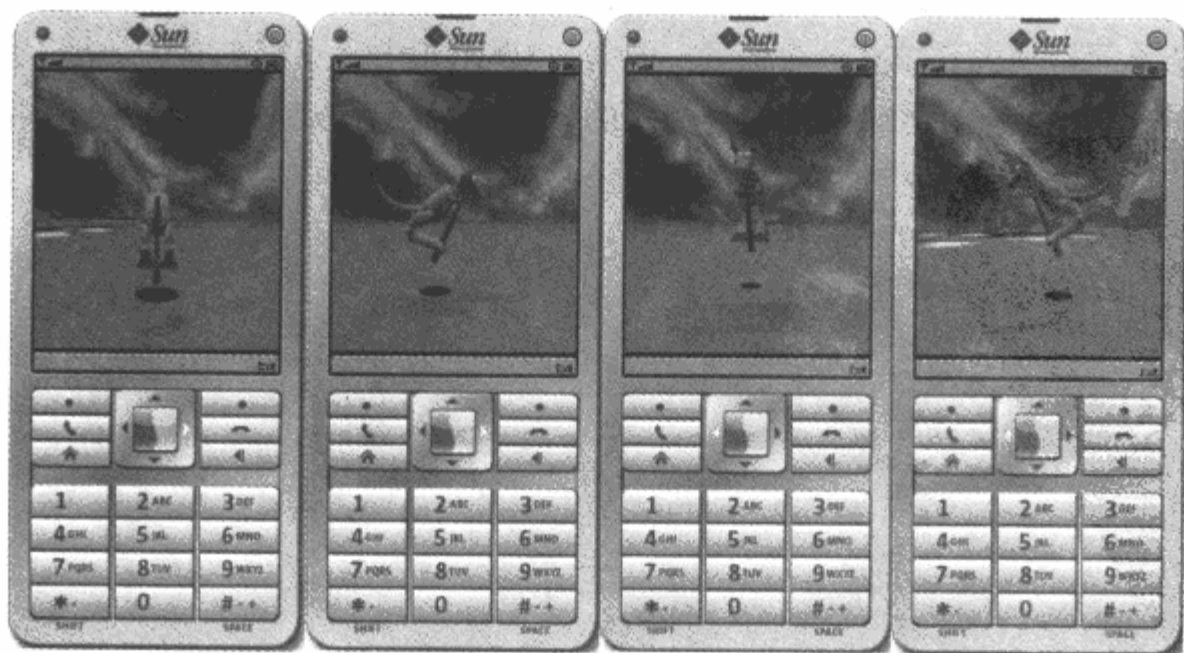


图 9-11 3D 游戏视角切换示例

3D 游戏的开发原理相对更复杂,其中需要涉及的内容非常多。在这里本书也就不再详细介绍,有兴趣的读者可以去网上下载 JSR-184 相关文档教程学习一下。

### 9.2.3 网络编程知识

手机一诞生就是应用于网络通信的,只不过当时的网络并不能进行除语音信号之外的传递。然而随着技术的发展,手机也开始替互联网承担一些网络服务,现在手机已经成为越来越多的人上网的首选设备了。

“师兄啊,我的这两个 Java ME 小游戏都没有开发联网功能,你说做手机游戏开发对网络编程知识要求高吗?”

“呵呵,说句实话,你这两个小游戏联不联网价值都不大。不过在实际的 Java ME 开发中,对联网的要求就比较高了。Sun 不是有句口号嘛:网络就是计算机。其实,手机网络也在不断壮大,也许不久以后这个口号就会改成:网络就是手机了呢。”

“看来到哪里编程也必须要和网络打打招呼啊。”

“Java ME 的应用,一般是游戏或作为企业级应用的客户端,后者如果没有联网功能就彻底没有了存在的价值。对于前者嘛,你想想看,开发了一款手机游戏,如果再基于联网技术开发出游戏的注册或购买模块,不就又多了一条生财之道嘛。”

不管面向的应用是什么类别,想要做一名合格的 Java ME 开发人员,就必须熟练掌握网络编程的知识。Sun 历来喜欢把基础性的工作做好让开发人员更多地把时间和精力用在对其他技术的追求上,Java ME 为开发人员提供了一个功能强大的网络连接框架:通用连接框架(Generic Connection Framework, GCF)。

“趁着这个机会,我给你讲讲 Java ME 开发中的 GCF 框架。”

“哦,我听过这个框架,正好没学习过,师兄你讲吧。”

“GCF 框架的最高层是 Connection 接口,其下设有 InputConnection、OutputConnection、StreamConnectionNotifier 和 DatagramConnection,这些子接口又会向下衍生,GCF 提供的网络传送方式有数据流方式和数据包方式,同时还可以把手机做成一个小服务器。”

“哦,功能的确不少啊!”

“不仅如此呢。MIDP 还对 GCF 进行了扩展,提供了很多基于协议的连接方式,如 HttpURLConnection 和 SocketConnection 接口等。”

除了好好利用 GCF 框架来进行高效的 Java ME 程序开发,本书还有一些手机网络编程的小知识拿来给读者分享。这些东西不算“歪招”,是经过实践总结出的小“秘诀”。作为 Java ME 的开发人员,这些都是很有价值的经验。

- 在开发 Java ME 应用程序时,如果需要连接网络,最好将网络相关代码写在一个独立线程当中,这样可以避免手机死机。
- 开发 cmnet 连接程序与 cmwap 的不同。cmwap 和 cmnet 是中国移动的两个上网接入点,主要的不同之处在于通过 cmwap 上网需要先访问移动的 wap 网关设置 HTTP 代理,一定程度上功能受限。而且在进行网络连接的开发时,如果手机使用 cmwap 接入点则有可能

需要两次或多次连接才可以成功，这是针对中国特殊网络情况的一种解决方案。

- 在前面也已经提过了不同软硬件平台对于 Java ME 的支持力度不同，在开发手机端网络应用的时候，最好在每次发送数据之后将发送缓冲区清空。因为有的手机平台只有在用户清空发送缓存的时候才会将缓冲区中的数据发送出去，如部分诺基亚手机平台就是这样。

#### 9.2.4 3G、Android 对 Java ME 开发人员的挑战和机遇

使用 Java ME 技术在手持式设备上应用开发，所需要面对的除了技术上的难关，还要关心的就是手机平台性能和网络带宽的问题了。就在本书完稿的时候，这两个需要关心的方面都出现了新生的技术，那就是 3G 和 Android。

3G 的到来，大大提高了手机和其他手持式设备的网络带宽，原来无法在手机上实现的许多应用，现在也可以流畅地实现了。以前有很多比较好的网络应用要求比较高的带宽，如在线视频播放、复杂的大型手机游戏等。在 2G 或 2.5G 时代，这些或者是太慢，或者都是不可能实现的。

“师兄，谈到手机开发，现在 3G 不是非常火吗？这对于 Java ME 的开发有什么影响吗？”

“从 2G 时代的 GSM，盼到 2.5G 时代的 GPRS，现在终于迎来了 3G。3G 的到来最大的特点就是带宽提高了，可以做的事就更多了，尤其是对于开发人员哦。”

“但是 3G 的价格……俺们老百姓啥时候能放心用啊。”

“3G 价格如果逐渐下降的话，用户将越来越多，面向 3G 的 Java 应用也将会越来越多。因为现在以及可以预见的将来在手机的上层应用领域，还是 Java 的天下。”

“看来我要准备好，万一哪天跳槽去搞 Java ME 了，呵呵。”

“呵呵，3G 一下子把带宽提高到一个新的层次，还有很多空白需要 Java ME 的开发人员去填补，这些可都是机遇啊。”

提到 Android，就不得不提开放手机联盟（Open Handset Alliance），这个组织是由 Google 倡导并于 2007 年成立的。Android 就是这个联盟共同开发出来的产物，这个联盟的成员遍布移动产业的各个领域，有移动运营商、半导体制造商、手机制造商、软件提供商等。当然，一些诸如苹果、RIM 等与 Android 有竞争关系的大公司并没有加入到这个联盟，中国移动也属于这个联盟。

Android 是 Google 推出的基于 Linux 平台的开源手机操作系统，对第三方软件开发完全开放。作为一款由互联网巨头谷歌牵头多家企业进行合力打造的新一代手机操作系统，Android 从研发之初就备受关注。

“既然谈到了 Java ME 在 3G 时代的发展，我们还得来提一提 Android 给 Java ME 带来的影响。”

“哦，是吗，那师兄你讲给我听吧。”

“Android 的推出肯定会对手机操作系统市场产生很大的冲击，也就会随之影响到 Java ME 的开发者了，Android 的内核当中就有很多是用 Java 开发的。Google 也已经高调宣布会大力支持 Java，估计 Android 会对 Java ME 有很大的扩充和提升，使其在移动开发的领域更加具有统治力。一些大的手机制造商如诺基亚也准备采用这个开源平台。”

“是啊，Google 都站出来了，看来前景不错啊。”

“Google 是那种屡出奇招的公司，想当年为了提升 Google Map 的用户体验，不遗余力地往天



上打了三颗卫星。最近几个月又和别人搞了一个‘O3b’网络计划，要发射16颗卫星以帮助非洲等地的国家享受到互联网。”

“的确，Google 成长这么快，和它总是采用新创意分不开啊。”

“呵呵，不管怎么说 Google 是个巨头，做事情靠谱，所以 Android 的前景不错。而且对应到国内，联想已经推出了第一款 OPhone 手机 O1，OPhone 手机是中国移动基于 Android 开发的面向国内手机的解决方案。OPhone 也提供了 SDK 和开发环境，你要转行的话可以去研究研究哦。”

随着 3G 时代的到来，以及 Android 操作系统的流行，原来在手机上功能受限的软件，现在可以尽情地增加功能，原来无法在手机上实现的应用，也可以放心大胆地实现了。3G 和 Android 分别在带宽和本地能力上增强了手机各方面的能力，使得手机越来越接近 PC。新技术的应用正在从各个方面改善着每个人的生活，如图 9-12 所示。

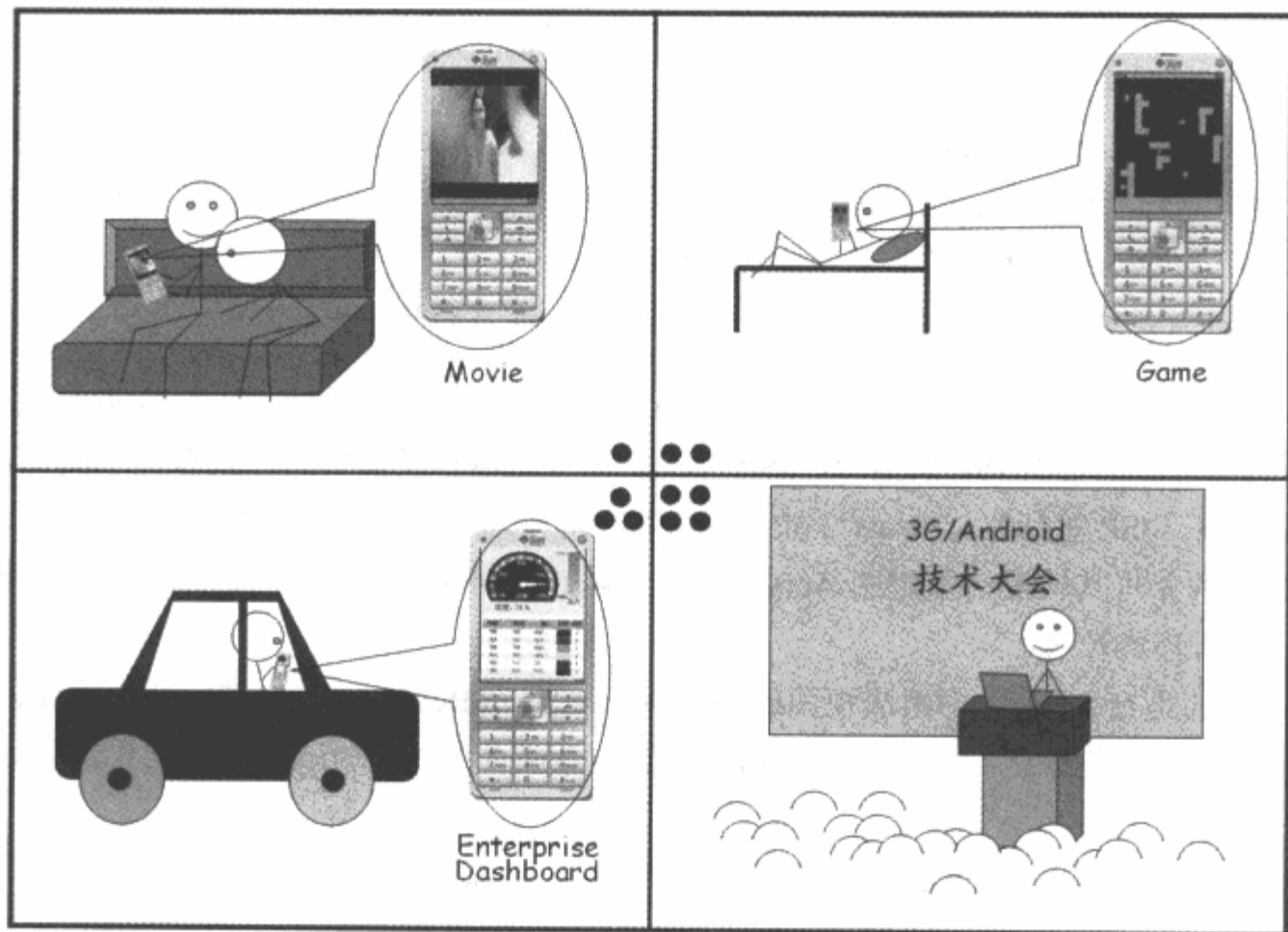


图 9-12 3G 和 Android 技术给手机应用带来的变化

以上只是 3G 和 Android 给人们生活带来变革的一个缩影，新的技术给用户带来全新的功能体验。而对于开发者而言，也是机遇大于挑战，作为一个合格的 Java ME 开发人员，了解并适当学习这些新技术，对于自己职业生涯的发展肯定是大有裨益的。

### 9.3 当下流行 EE 框架揭秘

9.1.2 节中介绍了身为 Java EE 开发人员所必知必会的知识，其中框架算是这个百宝囊中比较传统而又重要的部分，非常有必要深入探讨。本节将会介绍目前市面上比较流行的几种框架，讲述其工作原理和特性，希望能够对读者朋友的学习和使用有所帮助。

### 9.3.1 Struts 和 WebWork 那点事

Struts 和 WebWork 都是 Java 开源框架中元老级别的产品，本节将二者放到一起讲，是因为 Struts 和 WebWork 之间的确有着渊源。Struts 框架分为 Struts 1 和 Struts 2，下面首先介绍 Struts 1 框架的历史和工作原理。

“蔡佳娃，前天不是刚请我吃饭了吗？怎么今天又来星巴克请我喝咖啡啊，你没做什么对不起我的事吧？你这葫芦里卖的什么药啊？”

“师兄你看你想到哪去了啊！前天是真的有感而发地请师兄吃饭，不过，今天实不相瞒，我有个问题要请教师兄。”

“原来是这样啊，那是什么问题把我聪明的师弟给难住了啊？”

“哎，师兄你记得当初我学 Struts 框架的时候你让我重点学习 Struts 2 吗？”

“记得啊，你不会没听我的话吧？”

“没有没有，我是听了你的话，不过最近我发现我们公司做项目的时候用得最多的还是 Struts 1，我想跟我们经理说说换成 Struts 2，不过我对框架的了解实在太少，还要靠师兄帮我理一理这其中的思路，等到跟经理说的时候不至于关键时刻掉链子啊！”

“呵呵，原来是这样啊，那我们就来谈谈和 Struts 1 有关的故事吧。”

Struts 1 诞生于 2001 年，是 Java EE 框架中比较早的基于 MVC 设计模式的 Web 开发框架，在 Struts 1 中视图角色主要由 JSP 来担当，JSP 中没有业务逻辑和模型信息，只有用于浏览器窗口显示的 Struts 标签、JSP 等标签。Struts 1 中的控制器由一个核心组件 ActionServlet 来实现，它负责接收 HTTP 请求并将其转发给指定的 Action 对象。Action 对象负责调用模型的方法更新模型的状态和控制应用程序的流程。

需要注意的是在 Struts 1 中组成视图的还有 ActionForm Bean，它不像其他的 JavaBean 被用来处理业务逻辑，而是用来进行视图和控制器之间表单数据的传递。Struts 1 的工作流程如图 9-13 所示。

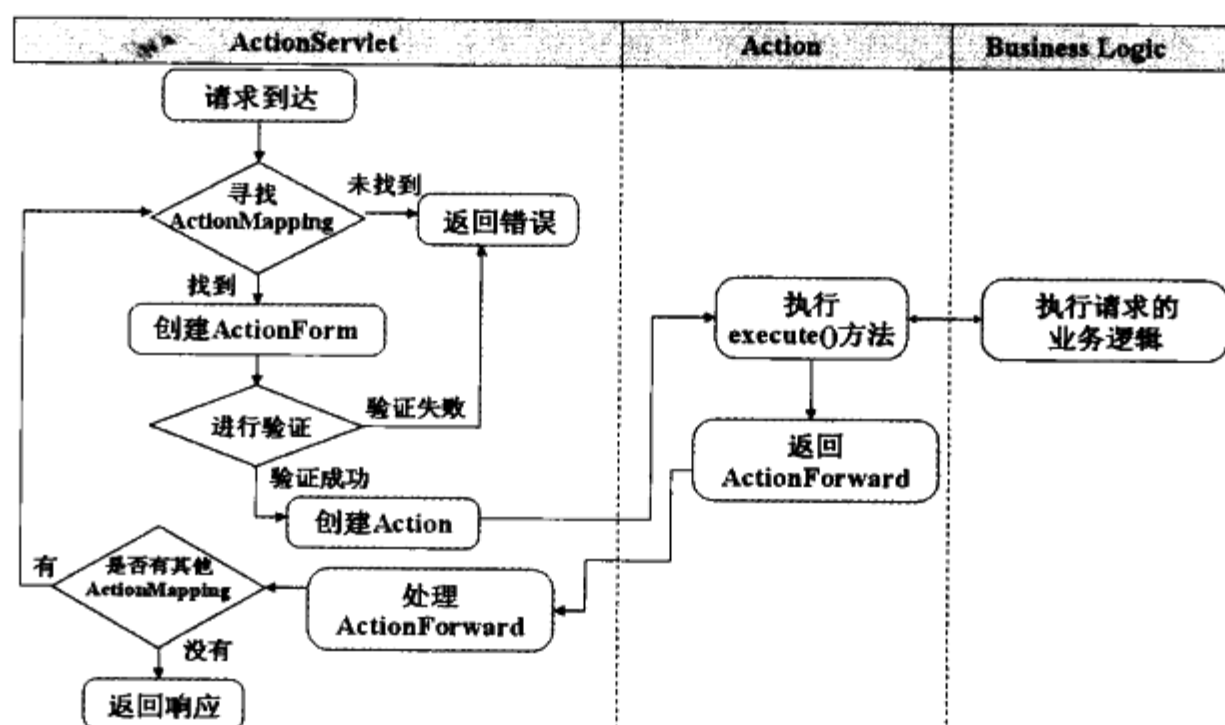


图 9-13 Struts 1 工作原理图

ActionServlet 接收到 HTTP 请求时，通过 ActionMapping 寻找需要调用的 Action 对象，找到后根据客户端浏览器发来的 ActionForm Bean 创建 Action Form 对象。再经过一系列的服务器验证，创建一个 Action 对象并执行其中的 execute 方法，该方法需要开发者重写，并有可能调用底层的业务逻辑。

execute 方法调用结束后返回一个 ActionForward 对象，该对象携带的是需要反馈给浏览器的具体视图，它将由 ActionServlet 处理并最终将响应返回到客户端浏览器呈现。

也许有的读者在看完上面的工作原理介绍后会觉得很麻烦，其实就是这样。使开发复杂化正是 Struts 的一个严重缺点。Struts 1 开发的不方便主要就体现在那个不厌其烦来回奔波的 ActionForm Bean 上，如图 9-14 所示。ActionForm Bean 在 Struts 1 的工作流程中强制设立了一些必须经过的关卡，使得本来简洁明了的 MVC 模式变得麻烦起来。

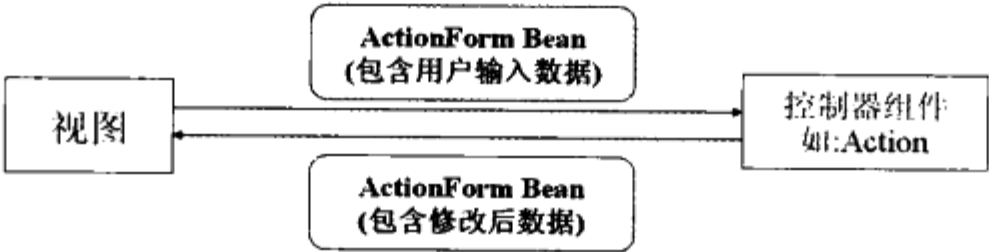


图 9-14 来回奔波的 ActionForm Bean

Struts 1 的另外一个缺陷就是其可扩展性相对较差，不利于开发人员在其基础上进行修改并增加新的功能。作为开源框架技术，修改源代码不失为一种扩展 Struts 1 的解决方法，但是其艰苦程度和难度系数可想而知。

“师兄，既然 Struts 1 的缺陷这么多，那它为啥还这么火呢？”

“原因就是 Struts 是最先问世的，Struts 1 一度可是开发人员的尚方宝剑呢，而其缺点也是在之后其他的框架不断涌现而对比出来的。这也是难免的嘛，最早的东西往往也是毛病最多的。而有了前车之鉴，后来的框架当然也就要比 Struts 1 优秀一些了。”

“哦，凡事都有个先入为主的概念嘛。”

“Struts 诞生不久，又诞生了 WebWork 框架。WebWork 这个框架确实比较出色，其实在国内对 Struts 追捧有加的时候，国外最火的还是 WebWork。但是从总体上看，WebWork 的用户还是没有 Struts 多。”

“WebWork 怎么和 Struts 扯到一起的呢？”

“Struts 名头大，历史悠久，但是思想更先进更好用的是 WebWork。随着技术的发展，WebWork 要推出 WebWork 2 了，而 Struts 虽然已经垂垂老矣，但是并没有搞出一个像样的第二版。在这种背景下，Struts 和 WebWork 的开发社区合并了，这样综合了 Struts 市场和 WebWork 技术的 Struts 2 就诞生了，这就是我为什么让你学 Struts 2 的理由了。”

合并后的 Struts 2 将完全取代原有的 Struts 和 WebWork 框架，WebWork 将不再推出新的版本。因此，原来在实际项目开发中使用 Struts 和 WebWork 框架的都将转而使用 Struts 2 框架。这样就将 WebWork 的开发者和 Struts 的开发都聚集到 Struts 2 的大旗之下。

Struts 2 对 MVC 模式的实现机制如图 9-15 所示，与 Struts 1 不同的是，Struts 2 支持的视图有

很多种，甚至可以用不同的技术去开发，如 FreeMarker、Velocity 等。Struts 2 的控制器由 FilterDispatcher 和 Action 来实现，这一点上和 Struts 1 是比较类似的。

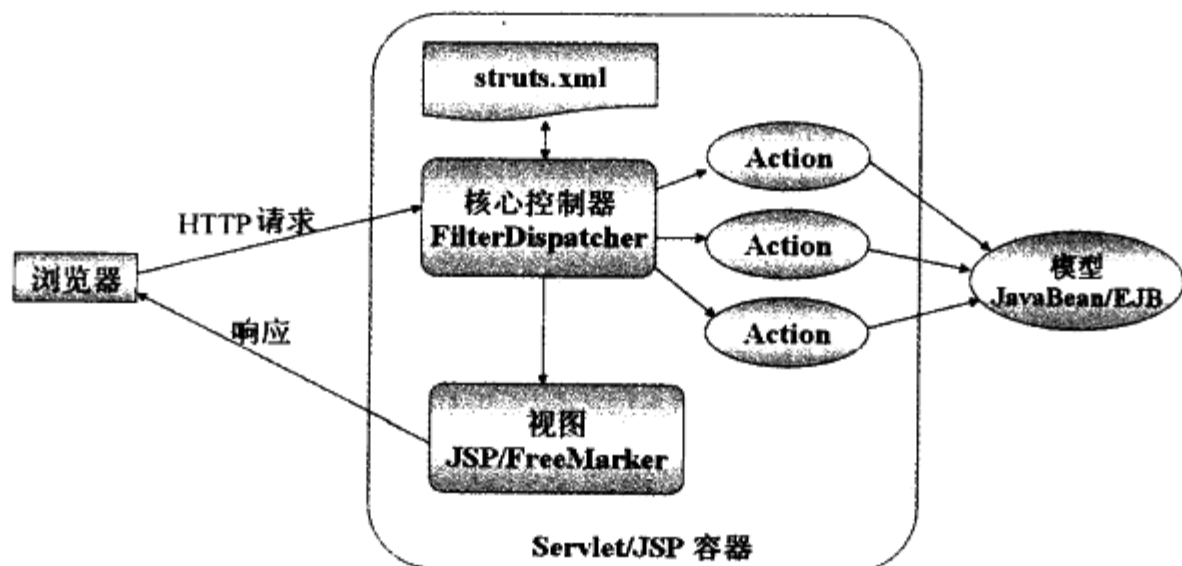


图 9-15 Struts 2 实现 MVC 机制示意图

相比较于 Struts 1，Struts 2 的一大亮点就是抛弃了 Struts 1 中的“鸡肋”——ActionForm Bean。Struts2 对 MVC 模式的诠释更加干净利落，HTTP 请求到达核心控制器之后，会经过处理直接找到 Action 对象执行相关代码，随后便选定一个视图返回浏览器。

“师兄，我看出来了，Struts 2 的确是很优秀，给人以精明干练的感觉。我想凭这个肯定可以说服我们老板‘改旗易帜’了。”

“不一定哦，只是方便了开发还不够，你得会研究老板的心理，老板们追求的是钱，即自己的产品是否好卖。”

“那我还有什么拿得出手的理由呢。”

“怎么会没有理由呢？Struts 2 的最大特性还没说呢，那就是支持系统级功能的热插拔。Struts 2 自带的核心系统功能本身就是一堆插件，可以轻松地拔掉后插上自己的插件，这样一来对框架进行扩展是极其方便的。”

“对啊，这个优势我都忘了呢。”

“只需要把 Struts 2 的核心部分看做一个可以插拔插件的底盘，开发插件的 API 对外开放，需要对框架进行修改时不需要进行源代码的修改，只需要开发插件，然后通过配置文件对功能进行组装、加新、去旧、替换就可以了。”

对于层次不太高的开发人员，只需要享受 Struts 2 带来的 MVC 实现机制的简洁明了即可；而对于层次比较高的开发人员，可以基于 Struts 2 平台开发自己的业务插件甚至是平台。

领略了 Struts 2 框架中 MVC 模式的风采，再学习一下 Struts 2 的工作原理会对其优良特性有更好的体验。如图 9-16 所示，在 HTTP 请求到来之后，首先经过 ActionContextCleanUp 等其他过滤器来到 FilterDispatcher，之后会来到 Action 映射器找到需要调用的 Action 对象，这里类似于 Struts 1，之后会通过 FilterDispatcher 来到 Action 代理，Action 代理通过调用配置管理器查找与所要调用的 Action 对象搭配的一系列拦截器，然后开始按照顺序调用这些拦截器和 Action 对象本身。

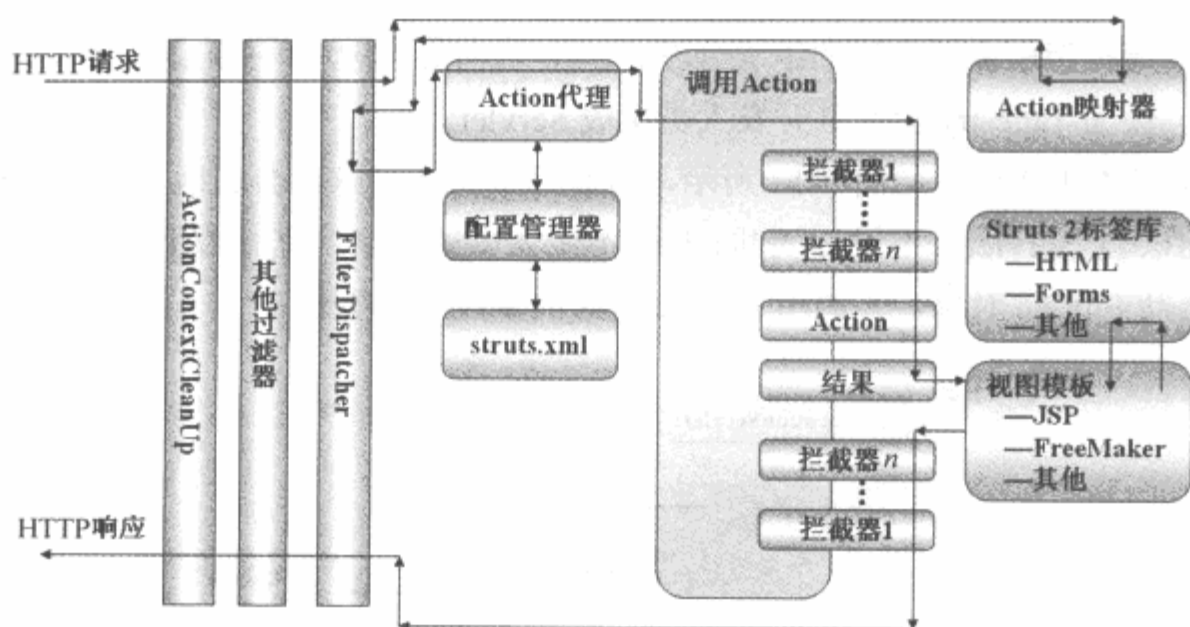


图 9-16 Struts 2 工作原理图

从图 9-16 中可以看出，过滤器和拦截器都是热插拔的，可以自己开发。例如原来系统日志都是记录在文本文件中，假设有一天需要将日志改成其他格式进行存放，如果以前把记录日志的功能写在 Action 里面的话，那么就必须要去修改 Action 的源代码，这样增加了许多工作量，代码的可维护性也较差。

Action 负责用户级功能，而日志、安全性、合法性检查等属于系统级功能，正确的做法是日志让一个拦截器去做就行，这样需要更改的时候只要更换一个新的拦截器就行了，业务开发人员只需要关注好用户级的功能即可。

### 9.3.2 Tapestry 框架

Tapestry 是另外一种开源的 Web 开发框架，虽然与 Struts 一样为 Web 层开发提供解决方案，但是 Tapestry 的理念却和 Struts 大有不同。不管是 Struts 1 还是 Struts 2 框架，所采用都的请求-响应这一传统的 Web 工作流程，而 Tapestry 与后面要介绍的 JSF，都颠覆了这个传统。

“蔡佳娃，既然说到了历史悠久的 Struts，我们不妨来谈一谈与 Struts 截然不同的一种 Web 开发框架——Tapestry。”

“啊，这个倒真是没听过诶，这个框架是做什么的啊？”

“先解释一下这个名词，Tapestry 在英文里是“挂毯”的意思，在 Tapestry 开发中的组件就像是挂毯一样通过不断地排列和复用达到 Web 页面的开发效果。”

“哦，这个说法倒是从来没听过啊，详细给我讲讲吧师兄。”

“好的，与以往不同，Tapestry 应用程序的构成元素是就是页面，页面是由组件构成的，组件还可以包含组件。组件是可以复用的对象，如按钮、文本框、表格等都是组件。”

“这些组件和我们传统意义上的文本框和表单有什么区别呢？”

Tapestry 中组件又被称为 JWC（Java Web Component），每个组件都会有一个 jwcid 以供开发人员调用。海量的组件库简化了 Web 表示层开发的难度，开发人员可以花更多的精力去做业务逻辑，而负责前端表示的人也可以在不会 JSP 的基础上使用组件做美工。

Tapestry 基于 Servlet，其应用程序需要运行在 Servlet 的容器中才行。Tapestry 最大的特点是



把开发人员从传统框架中需要关注的请求分派、URL 转向等问题中解脱出来，所以不了解 Servlet 的知识没关系，因为 Tapestry 会帮助开发人员生成 Servlet。

Tapestry 的基本工作原理如图 9-17 所示，所有的 HTTP 请求都会送达 `ApplicationServlet`，这是 Tapestry 的核心 Servlet。不过 `Application` 不做太多的实事，其接收到请求后并调用自己的 `service()`，以启动 `Engine` 对象，并调用 `Engine` 对象的 `service()` 方法。

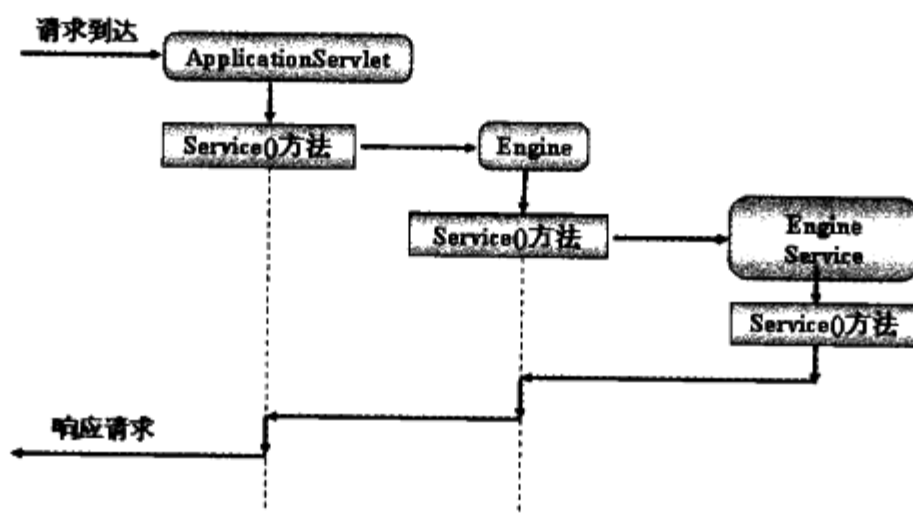


图 9-17 Tapestry 基本工作原理

`Engine` 对象是 Tapestry 框架中的枢纽，负责将所有的模块组织到一起。在 `Engine` 的 `service()` 方法中会进行很多工作，其中最主要的是调用 `Engine Service` 的 `service()` 方法。最后逐级返回，把响应返回给客户端。

`Engine Service` 也是一个 Servlet，确切地说是一个真干实事的 Servlet，在 Tapestry 中有 9 种不同的 service，如 `home`、`page`、`direct` 等。

使用 Tapestry 框架来做开发有很多好处，如下所列。

- 简单快捷，开发人员不需要关注底层，用现有或自定义的组件来构建应用程序即可。
- 真正的面向对象，而不是面向 Web 流程，Tapestry 的应用开发给人的感觉更像是开发桌面的 GUI 程序，Tapestry 中也有类似 GUI 程序的监听器。
- 错误报告比较详细。可以像普通 Java 程序那样精确到第几行。

Tapestry 的设计理念和其他框架不同，很多原理都与时下框架的工作原理大相径庭，所以在学习的时候可能没有学习其他框架快，但是学成之后就能体会到 Tapestry 框架的优越之处了。

### 9.3.3 Spring——不可多得的好框架

Spring 是一个优秀的 Java EE 开源框架，自 2004 年问世以来就一直受到开发者的好评。Spring 框架是一个比较全面的框架，可以为一些企业级开发提供“一条龙”式的服务，本小节将会对 Spring 框架的核心技术做一个介绍。

“蔡佳娃，上次你谋划的‘弹劾’Struts 1 事件进展得怎么样啊？”

“多亏了师兄啊，我们公司也已经决定将正在开发和准备开发的项目全部移植到 Struts 2 上了。不过既然如此，师兄你再给我讲讲别的框架吧，让我彻底做个 Java EE 的高人。”

“呵呵，也好，我们首先来看看 Spring 这个框架。Spring 框架是我用过的为数不多的好框架之一，今天针对 Spring 我们主要来谈一谈 IoC 技术和 AOP 技术。”

## 1. IoC 技术

IoC (Inversion of Control) 即控制反转技术, IoC 技术注重于面向于接口的编程, 即将对象与对象之间的依赖关系转变为对象和接口的关系。IoC 又被称为依赖注入 (Dependency Injection) 和好莱坞原则 (Hollywood Rules), 这些昵称都会随着对 IoC 技术介绍的加深而渐渐明朗起来。

“师兄, IoC 技术是怎么提出来的呢?”

“解决方案当然是问题产生之后才出现的啊, 在以往的开发中, 一个对象需要有指向另一个对象的引用, 才可以互相调用发生关联, 即产生依赖关系。这种对象之间的依赖关系是开发人员手动通过编程实现的。”

“对啊, 肯定要在代码上操作才可以实现啊。”

“手动编程有一个缺陷, 就是当对象之间的关系改变时, 随之而来的修改工作也大大增加。对象之间关系的改变有如下两种情况。”

- 在开发对象 A 时需要调用到对象 B 的功能, 假设 B 为 A 提供数据库访问功能, 由于项目的同时开发, 并没有成形的 B 类对象供调用, 此时的解决方案是写一个类来模拟 B 类的实现, 这个模拟类十分简单, 只要从 API 角度提供对象 B 的功能即可, 这样可以保证对象 A 的测试和项目的继续开发。但是到项目最后, 还需要修改代码将模拟类用真正的 B 类替代。
- 在项目的开发过程中写出了很多组件, 这些组件其实也是对象。而如果希望在部署时能够灵活地组装这些组件的话, 就必须退回去修改源代码进行对象之间的关联。而且在修改源代码之后, 必须要进行重新测试。

以上两种情况的解决方案都是需要手动修改代码, 实现起来很不方便。既然问题的根本是对象的开发和对象的依赖关系更改之间的矛盾, 那可不可以将二者彻底剥离开来, 开发对象的只管开发对象, 而对象之间的依赖关系由外部引入, 这就是 IoC 的开发模式。

IoC 的工作原理就是在开发一个对象的时候, 在关注其本身的功能之外, 只需要留出以后会调用的对象的引用即可。如果在对象的功能开发过程中必须出现所依赖对象的引用供调用测试时, 就开发实现相同接口的模拟对象来替代, 使得对象的功能开发可以继续进行。

当各个组件的功能开发完毕后, IoC 容器就开始发挥作用, IoC 容器决定某个对象保留的某个引用指向哪个具体的对象, 并将所有对象之间的依赖关系写入配置文件中。这样如果需要更改对象之间的依赖关系, 只需要改动配置文件即可, 免去了修改代码的复杂工作。

回到前面所提到的改变对象依赖关系的两种情况, 第一种情况下只需要开发一个替代类然后在部署的时候改动 IoC 配置文件将假的去掉换上真的对象即可; 而第二种情况下只需要对 IoC 配置文件进行合理规划, 即可灵活地完成各个组件之间的搭配。

“蔡佳娃, 明白了吧? Spring 的 IoC 是不是很好用啊?”

“实在是太棒了, 在 IoC 模式下开发, 各个组件的耦合度降到了最低, 而且开发完成后所有对象的关系完全由配置文件搞定, 修改起来实在是太方便了。”


“所以你明白为什么 IoC 也叫做依赖注入了吧, IoC 注入的依赖关系使得每个模块可以和其他模块共同完成工作。”

“嗯，那为什么叫好莱坞原则呢？”

“好莱坞的明星肯定忙呀，比方说你找他们推销点什么东西，他们一般会说：别找我，我需要的时候会去找你。这样就把这个买卖模式给反了过来，因为从来都是推销员找客户嘛。而 IoC 也是这样，原先都是对象在开发过程中决定要调用哪些对象，而 IoC 则改变了这个模式，一个对象应该调用哪个对象由 IoC 说了算，而不是对象本身了。”

“哦，我明白了，这个比喻果然很贴切啊！”

IoC 的工作原理颇似月老牵线，每个人就是所要关联的对象，这些人心中已经对自己的另一半有个大致的勾勒也为其留出了位置，但是并没有具体的人来填充这个位置。月老就是 IoC，当月老出现时，会根据自己的思想将他认为合适的男女进行牵线，这些被牵线的人之间就产生了关联，为心上人留出的位置也就有人来填充了。IoC 进行的就是一种“牵红线”的工作，只不过每个对象不会只和一个对象发生依赖关系罢了，配置文件大概就类似月老的“鸳鸯谱”。

 **提示** 依赖关系的更改实际上是在实现了相同接口的不同对象之中进行替换，这使得对象不必和其他对象绑定而是指向一个拥有很多实现类的接口，这样大大提高了灵活性。这种开发模式也就是前面提到的面向接口编程。

## 2. Spring AOP 的实现原理

面向方面的编程（AOP）技术的思想本书已经在前面的章节中介绍过，本部分将基于 Spring 框架，详细介绍 Spring AOP 的工作原理。Spring AOP 是 Spring 框架的重要组成部分，可以应用在 Java SE、Java ME、Java EE 的应用中，其代码 100% 是由 Java 开发完成的，因此具有很强的跨平台性。

“蔡佳娃，还记得我们之前提到过的面向方面的编程 AOP 吧？”

“嗯，记得。AOP 就是将散落在软件系统各个角落的如日志、安全性等系统级服务提取出来，集中放置在一个地方，谁需要谁就绑定给谁。这样就大大提高了代码的可重用性，同时当需要对系统级服务进行扩充和修改时，这种封装式的管理也使得系统易于维护。”

“嗯，说得不错，差不多就是这样。Spring AOP 是 AOP 技术的一种实现，我们现在就来细细讨论一下它的工作原理。”

要进行 Spring AOP 的开发，需要开发 AOP Device，即 AOP 装备，这是 AOP 开发中的主要内容，通常有 5 种装备类型。

- Before 装备，在执行目标操作之前执行的装备。
- Throws 装备，如果目标操作在执行的过程中抛出了异常，则该装备执行。
- After 装备，在执行目标操作之后执行的装备。
- Around 装备，在执行目标操作前后执行的装备，这种装备功能最强大，能够在目标操作执行前后实现特定的行为，使用最为灵活。
- Introduction 装备，此种装备能够为类新增方法，在五种装备中最复杂也最难掌握。

Spring AOP 的工作原理如图 9-18 所示，图中的客户端并不一定指终端用户，调用 Spring Bean 的代码也可以称为客户端。为了实现 AOP 思想，在 Spring AOP 中设置了一个代理工厂

(ProxyFactoryBean)，这是核心组件，相当于 Struts 2 中的 FilterDispatcher。业务接口以及实现了业务接口的 Spring Bean 需要注册到代理工厂，同时实现了装备接口（如 Before 装备接口）的实现类需要注册到对应的拦截器，拦截器再注册到代理工厂。

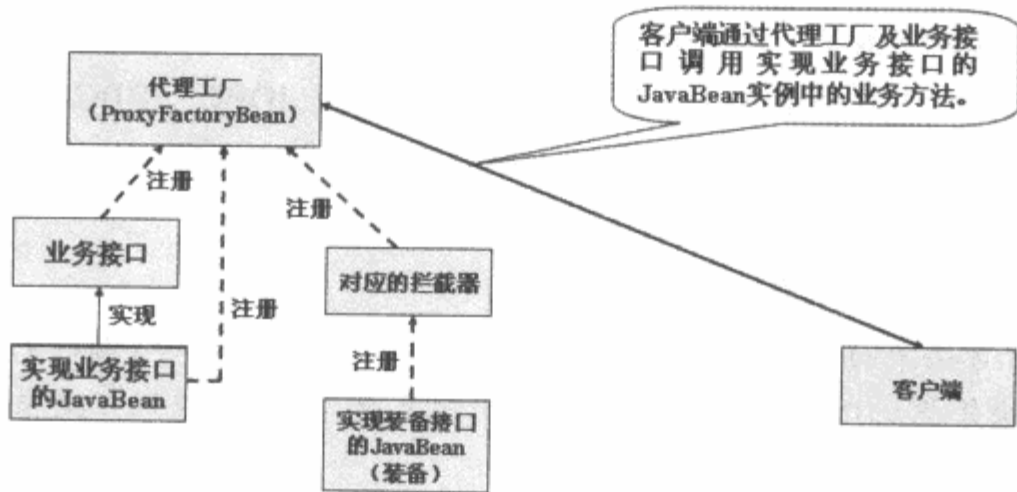


图 9-18 Spring AOP 的工作原理

当客户端需要调用实现业务接口的 Spring Bean 时，并不会直接获取指定功能 Spring Bean 的引用，而是将请求告知代理工厂，代理工厂通过查找配置文件，找到目标 Spring Bean 并根据其有无拦截器、是何种拦截器进行相关装备和业务功能的调用。

这样一来负责开发业务 Spring Bean 的开发人员就不需要关注这些横切关注点，只要根据实际情况决定业务功能需要什么样的装备即可，而开发装备的人只需要根据系统的要求对不同装备进行开发，最后对业务功能和系统级功能进行组装即可，如合法性检查需要放在 Before 装备中，而系统日志和出错日志需要放在 After 装备和 Throws 装备中等。

“师兄，Spring AOP 果然很高妙啊，写好一个装备挂到哪里，哪里就有了这种系统级功能，真是做到了‘一次开发，到处运行’啊。”

“是啊，而且开发人员还可以通过 Spring AOP 框架进行各种满足业务需求的自定义开发。例如当系统提供的安全性无法满足业务的需求时，可以自定义开发拦截器。”

“这个有点像 Struts 2 中的热插拔效果吧。”

“没错，这种对于二次开发的支持是每个优秀框架都应该具备的。其实提到 Struts 2，其中大量存在的拦截器和过滤器其实也在某种意义上实现了 AOP 的编程思想。”

“嗯，我也这么觉得。”

“最后，Spring 是一款非常出色的框架，其代码也非常优雅，如果你打算通过读源代码提高自己，Spring 应该是一个比较明智的选择。”

### 9.3.4 Hibernate——从关系世界到对象世界

谈 Hibernate 之前需要介绍一下 ORM 和 POJO。ORM 即对象关系映射 (Object Relations Mapping)，ORM 工具将关系数据库中的数据通过某种方式映射成对象。而 POJO 即 Plain Old Java Object，表示简单普通的 Java 对象。很多初学者最开始经常编写的 Car 对象、Student 对象等都属于 POJO。

Hibernate 是一个 ORM 工具，它可以将数据库中的数据映射为一个或多个 POJO，进而将面向

关系数据库的各种业务操作以 POJO 的属性与方法的形式实现。从而避免开发烦琐的 JDBC 代码，将精力集中在业务的实现上，这是 Hibernate 最大的一个优势。

“蔡佳娃，最后我们来谈谈市面上流行的另一种框架 Hibernate。”

“这个框架我用得倒是不多，师兄你给我讲讲吧。”

“Hibernate 的特点就是在与数据库进行关联的时候跳过了 JDBC。JDBC 是 Java 语言的数据库接口，其效率高、性能好。通过将数据库编程的代码抽象于不同类型的数据库产品之上并形成统一的 API，使得 Java 的数据库开发能力大大提高。”

“是啊，这么好的东西为啥还要抛弃呢？”

“由于 JDBC 是直接面向数据库查询语言 SQL 的，随着应用系统规模的扩大，数据库中表与表之间的关系越来越复杂，JDBC 代码也将变得越来越复杂，显然无法胜任关系型数据和面向对象编程之间的纽带工作。”

JDBC 的工作机制如图 9-19 所示，在应用系统规模很大的情况下很多时候数据库的性能和可靠性主要依赖于程序员的经验和技巧，这对于快速、大规模地开发软件系统是十分不利的。

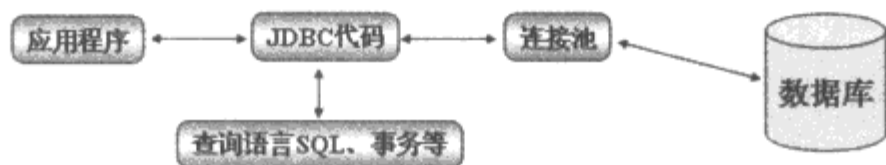


图 9-19 JDBC 工作机制图

而 Hibernate 可以掩盖掉复杂的 JDBC 过程，取而代之的是将 OR 映射后的结果展现在开发者面前，其余需要经验和技巧的设计部分封装在 POJO 对象和 XML 映射配置文件中。

Hibernate 的工作机制如图 9-20 所示，在 Hibernate 框架中，当需要调用数据库的时候，开发人员调用 Hibernate 的 API。由 Hibernate 负责进行与关系型数据库的交互，随后以 POJO 对象的方式返回给开发人员。如应用程序从学生数据库中查询成绩优秀者，Hibernate 会将每个符合条件的学生信息封装成为一个 POJO 类，并将其返回。这样对开发人员来说就不用将精力花费在数据库的操作上了。

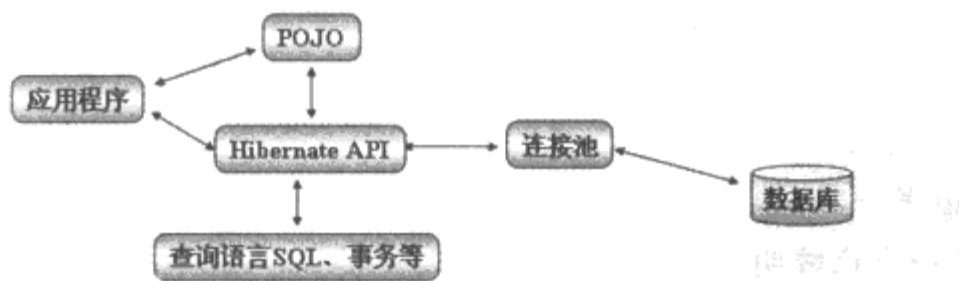


图 9-20 Hibernate 工作机制图

“数据库里面的元素都是关系型的，没有对象，而我们整天编程用的都是对象。使用 Hibernate 之后，业务代码就不需要直接使用 JDBC+SQL 操作数据库了。”

“是啊，这样关系型数据库的操作对于开发人员来说就变成透明的了。”

“正解，Hibernate 作为一种 ORM 工具，将面向关系的世界转化成面向对象的世界，这样把复杂的关系数据库操作带到了我们面向对象编程的一亩三分地上，极大地方便了开发。”



Hibernate 可以在各种 Java EE 服务器上面提供持久层的解决方案，也可以在桌面程序中直接利用 Hibernate 来完成对数据库的操作。同时 Hibernate 在支持集成方面提供对 JMX 标准的全面支持，实现了封装 Hibernate 全部功能的 MBean 接口。总之，在各种需要数据持久化的场景下使用 Hibernate 都是一种很不错的选择。

## 9.4 大型项目青睐的技术与平台

在前面一节中介绍的框架都有一个共同点，那就是全部都是开源免费的，而且那些框架并不属于 Java EE 的规范。这些框架技术在面对中小型应用的时候游刃有余，但在大型应用面前就有些捉襟见肘了。本节将介绍当前市面上大型项目所普遍采用的技术与平台。

### 9.4.1 JSF 框架

JSF 是 Java Server Faces 的简称，和 Tapestry 一样，JSF 也是基于组件技术的 MVC 开发框架，是一种区别于传统 Web 开发流程的技术。JSF 是由 Sun 推出的，因为 Sun 的名气很大，所以 Java EE 官方的 Web 层解决方案还是 JSF。而且和 Tapestry 基于 Servlet 不同的是，JSF 是基于 JSP 的。

需要特别注意的是，在 Java EE 5 中，JSF 只是一个标准，各个厂商和组织都可以根据标准的要求开发自己的实现，常用的 JSF 实现有 Sun JSF、MyFaces、Facelets JSF 等。

“蔡佳娃，除了我们刚刚谈过的 Tapestry 之外，还有一个颠覆传统 Web 开发流程的技术，那就是 JSF。这个比 Tapestry 的名气更大哦。”

“是吗？那师兄你给我讲讲这个 JSF 有什么特性吧。”

“首先需要提一下我们之前谈过的 IoC 技术，JSF 借助了 IoC 容器管理组件的依赖关系，使得单元测试和集成测试更加方便。”

“哦，JSF 很英明嘛，IoC 确实是个很好的设计思想。”

“不止这些呢，JSF 作为 Web 层框架技术，和 Spring 集成也特别方便。”

“我只是听过 Struts 和 Spring 集成，倒是没有听过 JSF 和 Spring。”

“JSF 和 Spring 集成起来要更方便一些，而且通过集成，可以将 JSF 简单的 IoC 和 Spring 功能强大的 IoC 同时挂起，这样一来 JSF 可以直接使用 Spring 的 IoC，就像使用自己的一样。”

用 JSF 开发 Web 应用的时候，将不会体会到请求-处理响应这个传统流程。与 Tapestry 相同，JSF 的主要组成部分也是页面，每个页面在后台都会有一个页面 Bean (Page Bean)，这个页面 Bean 会和前台的页面本身进行数据绑定。

比如前台页面包含一个用户登录组件，这个登录组件中肯定包含用户名和密码两个控件，于是后台的页面 Bean 中相应地就会有两个分别代表用户名和密码的字符串作为前台页面的数据模型。

当用户修改前台页面上控件变量的值并提交页面的时候，后台页面 Bean 的数据模型也会立刻随之改变。反之，后台数据的改变也会反映到前台的控件上，这使得 JSF 的 Web 开发更加类似于 Swing 等桌面程序的开发。

在一个 JSF 中页面的 form 一旦提交，JSF 框架要按照一定的顺序执行很多工作，其工作原理如图 9-21 所示。

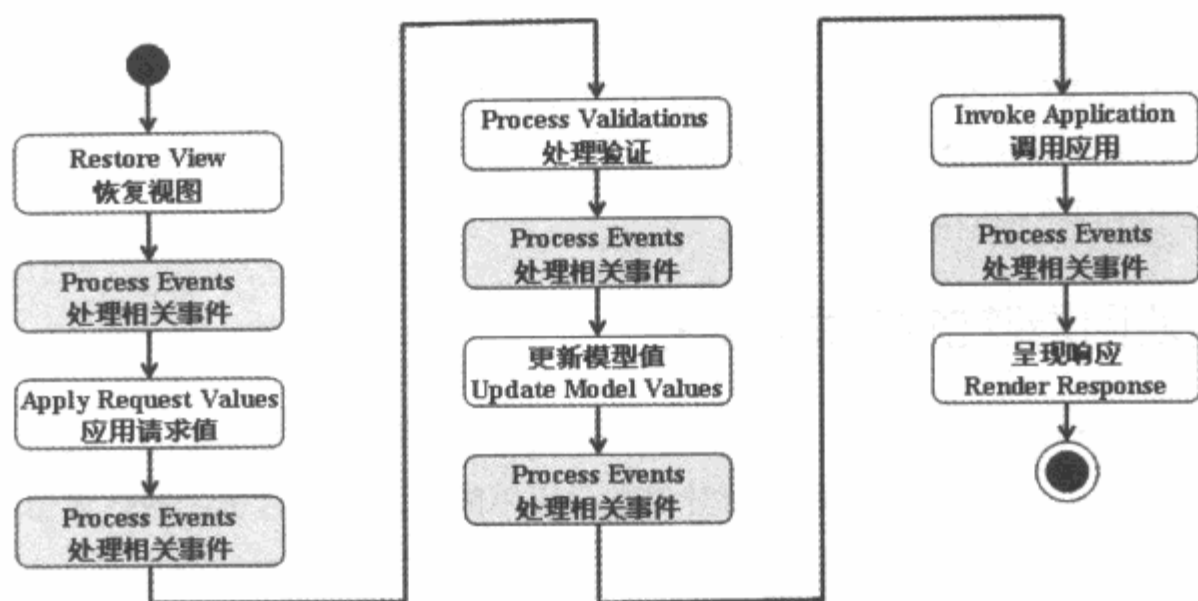


图 9-21 JSF 的请求处理过程图

为了便于读者理解，我们具体化一下这个请求，假设用户要更改自己账户中的年龄，根据图 9-21，这个请求每个步骤和相应触发的事件如下。

- 恢复视图

为选择的页面查找或创建一个组件树，恢复构成视图的组件，组件值可能来自用户传来的 request 或是存放在服务器端的数据。值得注意的是，有些组件会在此状态下产生动作事件对象。

对应到本例，就是在用户填写好修改年龄的信息并单击确定按钮提交 form 之后，后台需要将用户填写的修改年龄的页面组件重现，以便进行下一步。

- 应用请求值

恢复好的视图中每个组件各自从用户提交来的 request 对象中查找自己的值，并存储找到的值，同时还要对每个组件的值进行语法验证。

对应到本例，就是从 request 中提取到的用户名、年龄等值再重新赋值给恢复好的组件。语法验证区别于后面的语义验证，如本例中会检查输入的年龄值是否为整数。

- 处理验证

此阶段检查请求中的新值是否满足要求，如果出现异常将会抛出异常消息。对应到本例，就是检查用户提交的年龄值是否符合实际情况，如不能为负数或太大的正整数。

- 更新模型值

将通过验证的组件值更新到绑定的模型对象属性，即根据该页面组件值修改其页面 Bean 中对应变量的值。对应到本例，则是修改后台页面 Bean 的用户名和年龄属性。

- 调用应用

调用注册的监听器（如 ActionListener）来对模型对象的新值进行处理，对应到本例，就是修改数据库，把用户的年龄替换为新的数值。

- 呈现响应

针对先前收到的 request，根据导航逻辑返回一个合适的视图给客户端。对应到本例，则是返回提示修改成功或失败的页面组件给客户端浏览器。

“蔡佳娃，听了这么多 JSF 的介绍，你应该也感受到 JSF 和同为 Web 层技术的 Struts 2 框架之间的区别了吧？说说看。”

“嗯，我想想啊，Struts 2 的处理流程还是基于传统的请求-响应模型的。用户通过单击‘提交’按钮提交请求，请求被系统的控制器处理。这种模型对于传统的 Web 开发人员更为熟悉，使用起来也很方便，但是在使用 JSF 开发的时候就必须使用全新的思维方式来考虑问题了。”

“没错，因为 JSF 组件和标签的封装程度非常高，在典型的应用中已经不需要开发者去处理 HTTP 细节了。页面操作会被自动映射到后台的 JavaBean 中，处理逻辑直接与后台的 JavaBean 交互。”

“对啊，要不怎么说 JSF 和 Tapestry 更像 Swing 程序呢。”

“还有一个区别就是 JSF 使用 POJO 作为控制器，因而比较灵活，可以使用任意的方法来处理用户的请求。但是 Struts 2 的控制器必须是实现了 Action 接口的类或继承自 ActionSupport 类，其处理用户请求的代码只能写到 execute() 方法中。”

### 9.4.2 EJB 3.0 业务层技术

EJB 是 Enterprise Java Bean 的简称，同 JSF 一样，EJB 是 Java EE 的规范。EJB 工作在业务层，这点和 Spring 一样，但是 EJB 比 Spring 优越的地方在于其具有分布式能力，可以远程调用。EJB 是面向巨大业务量的重量级解决方案，而 Spring 只是轻量级框架技术。EJB 必须部署在 EJB 的容器中如 Weblogic、JBoss 等，目前 EJB 的最新版本为 EJB 3.0。

“师兄，除了 JSF，Java EE 还有什么规范吗？”

“有啊，其中一个很重要的就是 EJB 规范。作为 Java EE 在大型项目业务层上的官方解决方案，EJB 凭着其分布式能力和远程调用两个特性成为了这个领域最有力的武器。”

“哦，原来 EJB 这么厉害啊，不过我们公司规模还比较小，从我上班到现在还没接到过能够配得上 EJB 的大项目。”

“EJB 的结构特点使得开发人员在编写分布式应用程序时，不再需要关注事务、安全、多线程等问题，只需要在开发业务逻辑和部署这两个方面下工夫就行了。”

EJB 3.0 的分类如图 9-22 所示。在以前的版本中，还会有一个 Entity Bean，但是其后来被 JPA 技术所取代。EJB 3.0 主要分为会话（Session）Bean 和消息驱动（Message-Driven）Bean。前者主要负责同步业务的功能开发，而后者专注于异步业务。会话 Bean 又可以分为有状态的会话 Bean（Stateful Session Bean）和无状态的会话 Bean（Stateless Session Bean）。

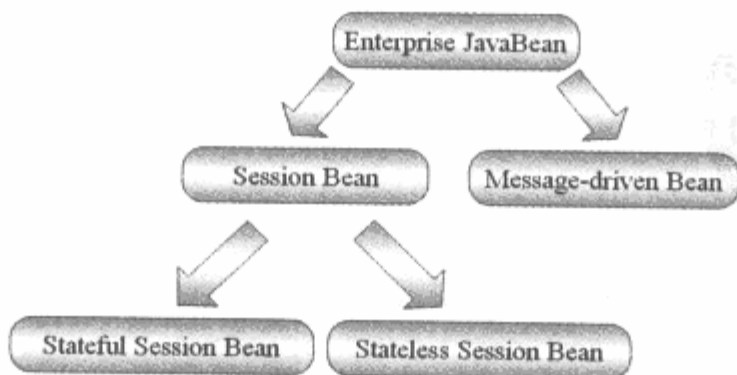


图 9-22 EJB 3.0 分类示意图

尽管 EJB 师出名门，但是 EJB 2.x 推出的时候，由于其开发的复杂性，使得大量用户将目光转而投向了 Spring、Hibernate 等轻量级框架。只是在 EJB 3.0 问世后才真正将 EJB 开发彻底“平

民化”，这样才重新聚集起了大量的客户及开发者。本书将对 EJB 2.x 和 EJB 3.0 的开发做一个对比，以此来加深大家对 EJB 3.0 的印象。

“蔡佳娃，EJB 的火爆，其实也是 EJB 3.0 推出后才有的事，之前的 EJB 2.x 可是许多开发人员谈之色变的技术呢。”

“啊，有这么严重吗？”

“你是不知道啊，当初开发一个最简单的‘Hello World’EJB，一个成手，没有半个小时也是搞不定的，更别说菜鸟了。”

“EJB2.x 的开发有那么复杂吗？这么复杂的技术干吗还要推出啊？”

“正当人们对 EJB 越来越失望的时候，诞生了 EJB 3.0。较之以前版本，EJB 3.0 带来的最大变化就是极大地简化了开发。”

EJB 的基础是 RMI，即 Remote Method Invoke，远程方法调用。EJB 3.0 的开发讲究的是“POJO+POJI”，POJO 前面已有介绍，POJI 是指 Plain Old Java Interface。“POJO+POJI”的意思就是用普通的 Java 对象和接口开发 EJB。

下面将分别采用 EJB 2.x 和 EJB 3.0 开发一个简单的“Hello World”EJB 类来体会一下 EJB 3.0 在开发中的便捷。首先是 EJB 3.0 的开发过程。

### 1. 定义远程接口

一个最简单的 EJB 类需要定义一个远程接口，在 EJB 3.0 中，定义一个远程接口的代码如下所示。

```
1 package ejbtest;
2 import javax.ejb.*; //引入需要的支持包
3 @Remote //说明此接口为远程接口的程序注解
4 public interface HelloRemote{
5     public String sayHello(); //业务方法的声明
6 }
```


这段代码除了“@Remote”，估计稍微学过 Java 的人都能看懂这是一个接口的定义。“@Remote”是程序注解，表示所定义的接口属于远程接口。

### 2. 开发 Bean 类

定义好远程接口之后，还需要开发一个实现了远程接口的 Bean 类，这样才可以通过本地或远程方式调用 EJB，在 EJB3.0 中一个 Bean 类的定义如下所示。

```
1 package ejbtest;
2 import javax.ejb.*; //引入需要的支持包
3 @Stateless(mappedName="HelloBeanJNDI") //用程序注解说明是无状态会话 Bean
4 public class HelloBean implements HelloRemote{
5     public String sayHello(){ //业务方法的实现
6         return "欢迎来到 EJB 世界! ";
7     }
8 }
```

定义 Bean 类的代码除了程序注解那一行，其他地方都非常容易理解。“@Stateless (mapped Name="HelloBeanJNDI")”中的“Stateless”表示定义的 Bean 是一个无状态会话 Bean，后面的赋值语句表明该 EJB 要对外公布的 JNDI 名称。

 **提示** 程序注解 (Annotation) 是为程序增加的描述信息，这些信息将会在编译、运行环境中发挥作用，可以简化代码开发，避免错误。

从上述内容中可以看出，在 EJB 3.0 中开发 EJB 是非常简单的，只需要结合程序注解写好相应的接口声明和类声明即可。而这些接口与类都是 POJI 与 POJO，开发没有很大难度，这也大大降低了 EJB 开发的难度与成本。

以上是在 EJB 3.0 中开发一个“Hello World”EJB 的过程，现在来看看在 EJB 2.x 中要达到同样的效果需要做的工作。

### 1. 定义远程接口

与 EJB 3.0 相同，EJB 2.x 中也需要首先定义一个远程接口，其代码如下所示。

```
1 package ejbtest;
2 import javax.ejb.*;           //引入需要的支持包
3 import java.rmi.*;           //引入需要的支持包
4 public interface HelloRemote extends EJBObject {
5                                     //远程接口必须继承系统的 EJBObject 接口
6     public String sayHello() throws RemoteException;
7 }
```

这段代码看似与 EJB 3.0 对应步骤的代码差不多，但是需要注意的是定义的远程接口必须继承自一个系统接口 EJBObject，而且接口中的函数都必须声明为抛出 RemoteException 异常。

### 2. 定义 home 接口

定义完远程接口之后，还需要定义一个 home 接口，其代码如下所示。

```
1 package ejbtest;
2 import java.rmi.*;           //引入需要的支持包
3 import javax.ejb.*;         //引入需要的支持包
4 public interface HelloHome extends EJBHome {
5     /** 这个方法创建 EJB 对象. */
6     public HelloRemote create() throws CreateException, RemoteException;
7 }
```

EJB 2.x 中的 EJB 实例将会通过此接口中的 create 方法创建，这是 EJB 3.0 中所没有的。

### 3. 开发 Bean 类

EJB 3.0 中开发的 Bean 类实现自远程接口，而在 EJB 2.x 中，Bean 类继承自 SessionBean 接口（如果它是会话 Bean），其代码如下所示。

```
1 package ejbtest;
2 import javax.ejb.*;           //引入需要的支持包
3 public class HelloBean implements SessionBean { //实现 SessionBean 接口
```



```

4      public void ejbCreate() throws CreateException {}
                                           //此方法对应到 home 接口中的 create 方法
5      public void ejbRemove() {}
                                           //实例销毁方法
6      .....
7      public String sayHello() {
                                           //业务方法的实现
8          return "欢迎来到 EJB 世界! ";
9      }
10 }
```

由于开发的 Bean 类实现自 SessionBean（会话 Bean）接口，使得必须为不必要的方法进行空实现，很大程度上使开发变得烦琐。

#### 4. EJB 配置文件

在 EJB 2.x 中，每个 EJB 都必须在配置文件中进行说明，如指出 EJB 名称、home 接口、远程接口、EJB 类名、EJB 类型等，代码如下所示。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  .....
3      <display-name>HelloBean</display-name>    <!-- EJB 显示名称 -->
4      <ejb-name>Hello</ejb-name>                <!-- EJB 名称 -->
5      <home>ejbtest.HelloHome</home>            <!-- home 接口 -->
6      <remote>ejbtest.HelloRemote </remote>      <!-- 远程接口 -->
7      <ejb-class>ejbtest.HelloBean</ejb-class>    <!-- Bean 实现类 -->
8      <session-type>Stateless</session-type>     <!-- Bean 类型 -->
9      .....
10 </ejb-jar>
```

#### 5. EJB 容器配置文件

在 EJB 2.x 中，不仅要有上述统一的 EJB 配置文件，针对不同的 EJB 容器，还需要为其建立特定容器的配置文件，本例以 Weblogic 为例，代码如下所示。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN"
3  'http://www.bea.com/servers/wls700/dtd/weblogic-ejb-jar.dtd'>
4  <weblogic-ejb-jar>
5      <weblogic-enterprise-bean>
6          <ejb-name>Hello</ejb-name>                <!-- EJB 名称 -->
7          <jndi-name>HelloJndiName</jndi-name>        <!-- 映射的 JNDI 名称 -->
8      </weblogic-enterprise-bean>
9  </weblogic-ejb-jar>
```

通过对比 EJB 2.x 和 EJB 3.0 创建一个 Hello World EJB 的过程，可以看出 EJB 3.0 在开发方式的简化上显示出了绝对的优势。EJB 3.0 的开发步骤相比于 EJB 2.x “缩水”不少，而且就算是对于二者共有的开发步骤，EJB 3.0 的开发也进行了很大程度的简化。EJB 3.0 真正做到了平民化，让不同层次的开发人员都容易上手。

细心的读者可能会发现，EJB 2.x 与 EJB 3.0 不同的是，远程接口和 EJB 类并没有发生实现关系，二者之间的对应关系是由配置文件进行约束的。而配置文件只在部署的时候起作用，在开发

过程中让 EJB 类去实现远程接口中的方法，就只能靠开发人员的人为约束了。

### 9.4.3 JPA 持久层技术

JPA 的全称为 Java Persistence API，是 Java EE 5 中一套完整的持久层解决方案。JPA 与 Hibernate 同处在持久层，但 JPA 并不是 Hibernate 的替代品。因为 JPA 是一种规范，可以由很多厂商去实现，Hibernate 中现在也有 JPA 的实现。

JPA 的出现终结了 EJB 2.x 中性能很差的实体 Bean (Entity Bean)，现在 JPA 已经是 Java EE 持久层技术的不二选择。再加上 JSF、EJB 3.0，三者组成了 Java EE 5 官方的解决方案，同时也成为了大型项目普遍采用的一套完整的解决方案。

“蔡佳娃，既然我们把 JSF 和 EJB 都谈了，剩下的一个 JPA 也一块说说吧。这三个规范是 Java EE 5 中为 Web 层、业务层、持久层提供的官方解答。”

“正好嘛，我们来个彻底点的探索，呵呵。”

“先来说说 JPA 的特点，JPA 吸收了许多主流持久化框架的优点，如 Hibernate 的 ORM 工具等。同时 JPA 没有和 Java EE 的容器绑定，因此可以在 Java SE 的环境中使用并对其进行测试。另外 JPA 定义了服务提供者接口 (Service Provider Interface)，在不用修改实体代码的前提下，开发者可以使用不同的持久化提供者，只要它们能够提供持久化存储的解决方案就行。”

“什么是实体呢？和 EJB 2.x 中提到的实体 Bean 是同一个概念吗？”

实体不同于实体 Bean，二者是不同的对象，而且实体也不是实体 Bean 的后继产物，是一种全新的编程概念。在典型的多层企业级应用中，通常存在两种不同的对象。

- 业务逻辑组件

此类组件对象提供业务方法，能够完成具体的业务逻辑，如根据单价计算用户订单、给客户的信用卡开账单等。在 EJB 3.0 中，这类对象一般由会话 Bean 来实现。

- 持久化数据对象

通过不同的持久化机制能够将 Java 对象存储到持久化源中，这类对象用来表示数据，例如银行账户信息、人力资源数据等。在 JPA 中，这些被持久化的对象称为实体。

JPA 推崇 POJO 的编程模型，所以在 JPA 中，实体都是 POJO，开发者能够将其持久化到持久化源（数据库或遗留系统）中。实体中以属性的方式存储数据，可以通过方法关联到相应的属性。

“原来实体就是 POJO 啊，那开发实体类和其他类有什么不一样的地方吗？”

“要特别注意的是，实体必须声明主键，这点和数据库表差不多。同时实体中不但可以有存取属性的方法，还可以有一定的业务方法，如扣除账户余额、向账户内存入资金等。”

“看来实体在 OR 映射中发挥的作用还真不小呢。”

“另外，你还记得上次我们提到的 Annotation 吧，实体类中也可以通过注解的方式给出生命周期的回调方法。”

最后要注意的是，JPA 中的实体是本地对象，不能够直接对其进行远程访问。如果需要远程访问，一般是采用会话 Bean 对其进行包装，在 JPA 规范中实体的工作步骤如下所列。

- 从持久化源中装载数据，并提供属性域来存储载入的数据。

- 通过修改内存中的 Java 对象，改变数据的取值。
- 将数据存回到持久化源中，从而达到更新持久化数据的目的。

#### 9.4.4 常见应用服务器简介

俗话说：“好马配好鞍”，前面介绍的都是针对企业级应用不同层次的 Java EE 官方或开源解决方案，用这些技术开发出的应用只能算是“好马”，要想让用户驾驭好这匹“好马”，必须得配上“好鞍”。这些“好鞍”就是应用服务器，提到应用服务器或许很多人会想到 Tomcat。虽然 Tomcat 很出色，但是并不能满足高级的应用，本节将简单介绍市面上流行的几种企业级应用服务器。

##### 1. Weblogic 应用服务器

Weblogic 的 logo 如图 9-23 所示。Weblogic 是 BEA 公司推出的一款 Java EE 应用服务器，Weblogic 并不是 BEA 公司的原创，而是其收购来经过加工和扩展开发出来的。Weblogic 目前在市场上占有的比例较大。



图 9-23 Weblogic 的 logo

Weblogic 优于其他应用服务器的地方就是集群技术，它实现了 Web 集群和 EJB 组件的集群，这些都大大增强了系统的可扩展性和高可用性。

##### 2. WebSphere 应用服务器

WebSphere 的 logo 如图 9-24 所示，它是 IBM 遵照如 Java EE、XML 等开放的标准推出的集成软件平台。WebSphere 在 1998 年发布的时候只能算是个 Servlet 容器，经过十年左右的发展和完善，WebSphere 已经成长为大型企业级项目首选的高级应用服务器之一。



图 9-24 WebSphere 的 logo

WebSphere 在用作电子商务平台的时候显现出了很大的优势，WebSphere 有三个版本，从简单的类似 Tomcat 般的 Web 容器，到支持实现了 CORBA 等技术的高级应用都能轻松胜任。

##### 3. JBoss 应用服务器

JBoss 的 logo 如图 9-25 所示，JBoss 是一个基于 Java EE 规范的开源应用服务器。与前面两个应用服务器最大的不同就是使用免费，所以很受企业和教学单位的青睐。



图 9-25 JBoss 的 logo

JBoss 的开发集中了全球开发者的优秀思想，JBoss 主要作为 EJB 容器而存在，当需要支持 Web 级应用时，则应和其他的服务器如 Tomcat 等联合使用。与其他应用服务器相比，JBoss 所占空间更小、易用性更好，同时对于最新的 Java EE 等规范支持也比较好。

### 4. GlassFish 应用服务器

GlassFish 的 logo 如图 9-26 所示，它是另一个开源免费的 Java EE 应用服务器。GlassFish 的前身是 Sun 公司的 Sun Java System Application Server PE 项目，后来 Sun 将其源代码开放，同时整合了 Oracle 捐献的 TopLink 代码，变成了现在的 GlassFish。



图 9-26 GlassFish 的 logo

有了 Sun 和 Oracle 的基础，GlassFish 在广大开发者的辛勤工作下快速成长，任何人都可以下载源代码并进行修改和发布。GlassFish 主要用来支持 Java EE 应用，但是它对 Web 应用的支持也很完善，并不逊色于 Tomcat。

## 9.4.5 Java 企业平台的荣耀之路

Java EE 作为 Java 技术中最重要的应用平台，在大型应用的项目中发挥着举足轻重的作用。同时 Java EE 也是成长比较快的技术，从 1998 年 EJB 1.0 的诞生到即将发布的 Java EE 6，Java EE 经过了多次变迁，涌现出了许许多多的新技术、新框架以及新应用服务器平台。

Java EE 的发展之路如图 9-27 所示，可以将 Java EE 的发展分为 Servlet 阶段、J2EE 阶段和 Java EE 阶段，下面就来向各位读者展开这部 Java EE 的荣耀史诗。

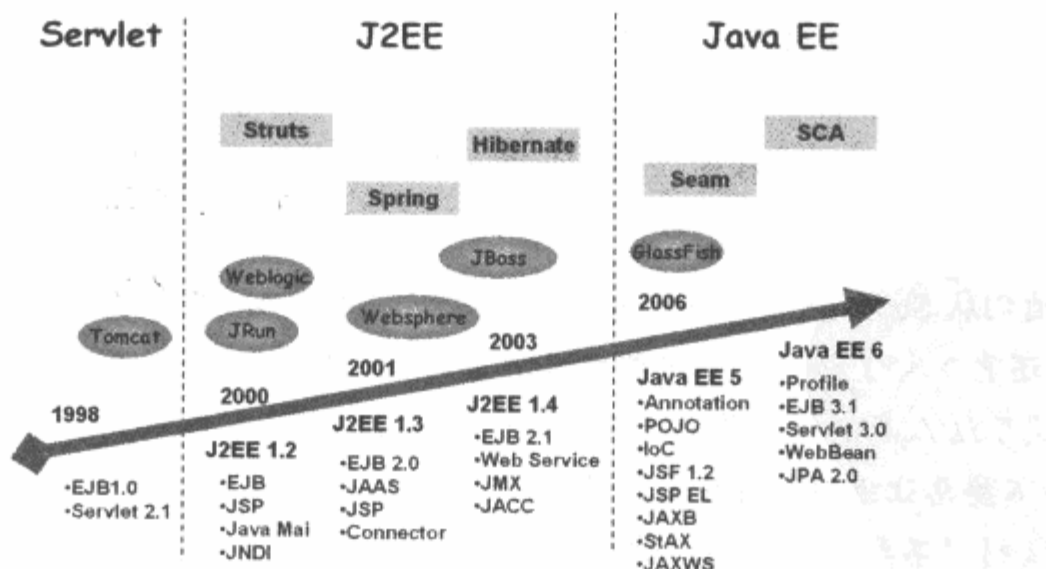


图 9-27 Java EE 荣耀之路

### 1. Servlet 阶段——初露端倪

Java 技术诞生于 1995 年，而于 1998 年发布的 EJB 1.0 则正式拉开了 Java 企业级开发的序幕。

在这个时期，Java 仍显得有些青涩，应用服务器也只有 Tomcat 来支持 Servlet。

## 2. J2EE 阶段——虎虎生威

这个时期是 Java 企业平台迅速成长的时期，Sun 相继推出了 J2EE 1.2、J2EE 1.3、J2EE 1.4 等规范，其中包含如 JSP、Java Mail、JMX 等技术。同时 EJB 也发展为 EJB 2.x，这些都极大地促进了 Java 在企业级开发中的应用程度。

同时，Struts、Spring、Hibernate 等如今依然活跃的框架也是在这个时期诞生的，而应用服务器方面 Tomcat 也不再孤单，WebSphere、Weblogic、JBoss 都为服务器平台注入了新鲜的血液。

## 3. Java EE 阶段——登峰造极

这个时期 Java 企业平台正式改名为 Java EE，在新技术规范的推出上依然强势，Java EE 5 除了推出了本书前面介绍过的 EJB 3.0、JPA 等技术，还对 SOA 中需要的 Web 服务开发提供了良好的支持。而 Java EE 6 则将万众期待的 Servlet 3.0 搬上舞台，同时还提出了 WebBean、JPA 2.0 等最新技术。

应用服务器方面，GlassFish 的加入壮大了开源服务器的力量，而新框架技术如 JBoss Seam、SCA（Service Component Architecture）等大大地丰富了 Java 在企业级开发中的解决方案。

# 9.5 如何学好框架

前面几节本书介绍了一些流行的开源框架和 Java EE 5 规范中企业级开发的解决方案，这些技术对于 Java EE 开发人员来说是非常有必要学习和掌握的。本节就来简要谈谈如何学好框架这个话题，希望对各位读者的学习提升有所帮助。

## 9.5.1 全面了解各项功能

相比于其他知识如核心 Java，框架技术的应用特征比较明显。例如学了 Java 的泛型这个特性，可能当下并不会用到，只是做了一个知识的储备。而学习一个框架技术，往往是直接面向应用的，例如不会有人研究完 Tapestry 之后将学到的知识“雪藏”起来不立刻使用的。

框架技术的这个特性，也使得很多人在学习的时候太注重框架的某个部件的使用而失去了对框架整体上的把握。

“师兄，咱们从 Struts 讲到 Hibernate，又从 JSF 讲到 JPA，有一些框架技术我是掌握了，有些像 Tapestry 还真不太了解，对于这些琳琅满目的框架技术，我们应该如何学习呢？”

“呵呵，谈了这么多的美味佳肴，也该谈谈如何烹饪了。以我之见，学习框架技术的时候，第一个原则就是不要总让框架给你带来惊喜。”

“嗯？什么叫‘不要总让框架给你带来惊喜’啊？”

“如果你总是听到一个人说诸如‘哇塞，Struts 2 还可以这么干！’或者‘咦，Spring 里没有这个功能吗？不应该啊！’之类的话，这就表示框架给这个人带来过多的惊喜了。”

“哦，原来是这样啊？这样的人有什么不好吗？”

“这样的人永远也不会真正地掌握一种框架技术，他们总是被动地、试探着去学习，总是等着



需要的时候再去学，这样对于整个框架总是不会有个全面的掌握。”

“对，学习就是应该主动啊。”

故事中提到的那种人就像是懒人晒太阳，本来站在太阳下，阴凉侵略过来了，就往有阳光的地方挪一挪，过会发现自己又享受不到阳光了，就再挪一挪。这样每次挪动都会浪费时间和精力，为何不一步到位，寻找一个能够保证自己在有限的时间内永远可以享受阳光的地方呢？

“所以说啊蔡佳娃，你在学习框架的时候不要抱着在大学时的‘六十分万岁，多一分白费’的思想去研究，我给你的建议就是从整体入手，直接研究 API。”

“啊，上来就这么整啊？”

“对，先把 API 研究透，看看这个框架到底能做什么，不能做什么，这样才能全面地了解框架技术，使你对它的认识更加有整体感。”

框架就像一个品种繁多的大果园，很多时候开发人员需要的只是一种或几种水果，但是最好不要因此而忽略了整片果园。要好好研究果园里都有什么水果、什么水果在这个果园里种出来最好吃、什么最难吃，这样才能更好地利用这片果园。在对框架进行深入研究或开发应用之前，彻底地了解框架的能力范畴才会有最多的收获。

### 9.5.2 彻底研究工作机理

学习框架，一方面是为了更好地使用框架，另一方面就是要体会框架的工作机理，并且在以后的开发中将框架技术的博大精深运用到自己的产品中。就像本书前面也曾经介绍过的，研究框架的工作机理或者是源代码也是一个开发人员从菜鸟走向牛人的可选途径。

“蔡佳娃，在使用框架的时候，除了要研究框架的 API，搞清楚什么能做什么不能做，还要试着掌握其工作机理，这样才能更加灵活地驾驭框架技术。”

“师兄，那掌握工作机理的途径都有什么呢？你不会又想让我去读源代码吧？”

“哈哈，你很聪明嘛，不过看起来你对于读源代码很抵触啊。”

“也不算抵触，只是觉得应该还有比这个稍微轻松点的方案吧？”

“的确，研究好框架的工作原理途径不止这一种，你可以在长期的使用开发中积累经验和技巧，逐步理解框架的内部机制。不过一种是站在外面猜，另一种是捅破窗户纸看个明白，两种方案的孰好孰坏就不用我多说了吧。”

“话倒是这么说，只是想到那么多代表高深思想的代码密密麻麻地浮在眼前，我就没有继续下去的勇气了。”

“没有试过怎么知道自己会不会呢？凡事都是从不会到会的，想想看以前学 Java 不也很困难吗？现在你都靠 Java 来养活自己了，还有什么做不来的呢？”

对于一名开发人员来说，在使用框架的时候，可以将其分为三种境界。第一种人能力很差，掌握的知识浮于表面的技术；第二种人能力很高，所有的框架实现细节都清楚明白；第三种人能力中等，对底层的知识有所了解，但是又没有学透。

对于前两种人来说，虽然境界相差甚远，但是至少都不会痛苦，能力差的人大不了高深的功能不研究不开发也不操心，而能力高的人就更没什么问题了。也就是第三种人最痛苦，这种人已

经超越了初级的水平，但是却总是苦于登天无门。如何才能越过这个阶段成为高人呢？最好的办法是改变自己的位置，从框架的使用者变成框架的研发者，批评地看待框架而不是迁就。

“对于学习框架技术来说，掌握工作机理的方法很多，读源代码并不是唯一的途径，但是遇到某种情况，就必须去啃源代码了。”

“这么说师兄你已经啃了不少了？”

“我啃源代码都是被逼的啊，有一次我用 Hibernate 3 做持久层开发，Hibernate 3 提供了事件监听器的新特性，我就用它来开发自己的监听器，挂上后我就开始测试，发现只有我自己的监听器工作了，而 Hibernate 的系统功能莫名其妙地蒸发了。”

“啊，是不是你把系统功能给关了呢？”

“当时我也是这么怀疑，但当时没有高人指点，网上也没有这种解答，无奈之下我只好去读源代码了。结果就发现 Hibernate 的系统功能本身就是用监听器实现的，挂上我自定义的监听器后系统监听器就下岗了。”

“哦，问题在这里啊。”

“找到问题后，解决起来就十分方便了，直接在自己的监听器中调用系统就行了。这下你明白读源代码的好处了吧，”

以上都是在探讨如何更好地使用一个框架，其实框架除了被用来开发使用之外，还是有很多存在价值的。框架之所以广泛被开发人员追捧和使用，其中必定有其高妙之处。研究工作机理和源代码不仅可以更好地使用框架，更重要的是可以学到其中的思想并为我所用。

## 9.6 本章小结

本章在前两节分别探讨了身为 Java EE 和 Java ME 开发人员所必须要掌握的知识 and 能力，读者在阅读时可以同时给自己做一做“体检”，找出自己不足的地方然后为之努力。

随后，本书介绍了当前市面上流行的框架技术及开发大型项目采用的技术与平台。在介绍这些知识的时候，着重讲解的是框架技术的工作原理和特性。对于读者朋友并不熟悉的框架技术，可以通过本书先“知其所以然”，在以后“知其然”的时候会更加事半功倍。

# 第 10 章 几种自废武功的做法

作为一名 Java 开发人员，正常情况下在工作的过程中会不断地增长知识，积累经验。但是如果采用了不恰当的方法，坚持了不该坚持的习惯，后果则是不但不会让自己在知识和技能方面有所提高，甚至会武功尽失，再也没有一技之长行走于江湖。本章就来揭示一些会让开发人员自废武功的做法和思想，希望各位读者朋友们不会在这些方面重蹈覆辙。

## 10.1 相信谬论

就像“地心说”曾经在人们心中顽固地存在了上千年一样，Java 开发界也存在着大家都信以为真的谬论。这些谬论或者只是在理论上称霸，从来没有人去实际验证；或仅仅是在特定历史时期曾经为真，而现在早已是另一番天地。本节主要将 Java 开发界最著名的谬论一一推翻。

### 10.1.1 说出来别不信——链表和数组的速度问题

自从和牛开复讨论过手机游戏的知识之后，蔡佳娃就迷上了开发小游戏，不过由于懒得抽时间将游戏部署到手机，他开发的都是些桌面小游戏。

“师兄啊，你还记得上次我给你发的那个小游戏代码吗？你看了没有哇？”

“哦，是那个‘蔡蔡连连看’吗？”

“嗯，我运行的时候总是比较卡，游戏体验很差啊。”

“我还没看完，不知道你的核心算法有没有问题，不过你可以先回去把所有的 LinkedList 改成 ArrayList 试试。”

“不对吧，师兄，我这个‘蔡蔡连连看’里的对象需要频繁地插入删除，频繁插入删除不是用链表比较好吗？怎么还让我改成 ArrayList 啊，那不更慢吗？”

“谬论，这是谬论啊，看来我还是赶紧给你灌输正确的思想吧。”

搞软件开发的几乎没有没学过数据结构的，学过数据结构的没有没学过数组和链表的，学过数组和链表的没有不把二者作比较的，作比较的没有不得出这样一个结论的：数组在随机存取方面要比链表快，而链表在处理节点的频繁插入删除时性能要优于数组。

这个结论的前半部分本书是认可的，数组有按照索引值随机访问的能力，效率当然比链表的顺藤摸瓜要高。但是结论的后半部分就是谬论了，本小节将对其进行抽丝剥茧式的探讨。

#### 1. 谬论的产生

首先来分析这个错误论断的产生，数组在计算机中的存储是一块连续的内存，通过索引访问；而链表则是不连续的存储单元，通过指针关联，这个基本知识想必大家都明白。数组中执行插入操作时，将插入点之后的元素依次后移一位，然后将新元素插入到腾出的位置。而进行删除操作时，则是将指定位置的元素删掉，之后的元素依次前移一位，数组的插入和删除操作原理如图 10-1 所示。

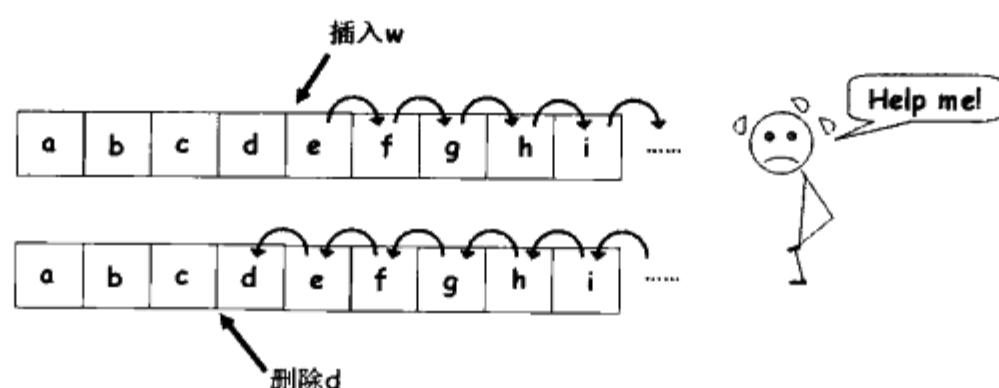


图 10-1 数组插入删除元素示意图

这看起来着实是一个比较烦琐的工程，实际项目中的数组长度短则几个元素，长则数万个元素，假设要在第一个节点处插入或删除元素，就必须将动辄上万个元素依次挪动一遍。

而对于链表，处理方式就很简单了。需要在某个节点之后插入节点的时候，只需要将新节点的后继指向该节点原本的后继，再将该节点的后继指向新节点即可。删除节点则更为简单，只需要让被删节点的前驱将后继指针指向被删节点的后继即可。链表的插入和删除操作原理如图 10-2 所示。

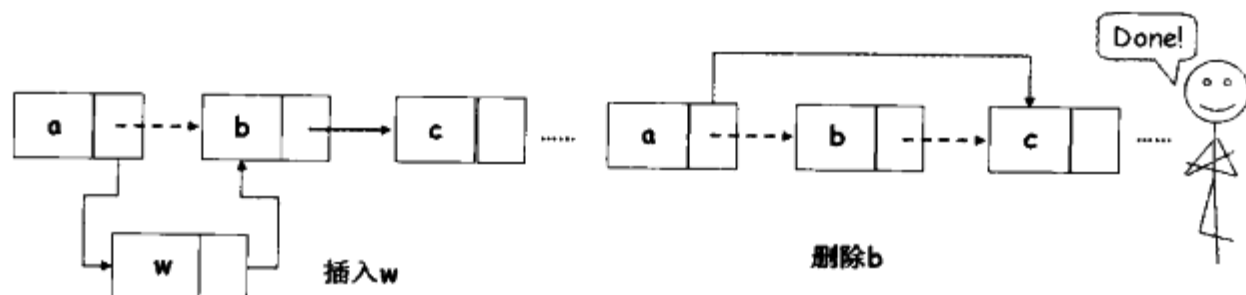


图 10-2 链表插入删除元素节点示意图

稍稍思考一下，所有人都会认为数组在这方面效率很差，笔者也承认是这样。但是这个比较从根本上来讲就是不公平的，无论是链表还是数组，插入和删除操作都需要两个步骤：找到位置和进行具体操作，上面的比较只是对比了执行具体操作的效率，忽略了找到位置所花的时间。

对于数组来说，查找指定位置就是计算索引，这在计算机中的实现时间是可以忽略不计的，主要的时间将花费在进行具体操作上。而对于链表来说，找到指定的位置时，整个操作就基本上完成了。因为对链表而言执行具体操作的时间与寻找指定位置相比，几乎可以不用计算在内。因此上面的比较中计算了数组操作的大头而只计算了链表操作的小头，这样的比较是明显有失公平的。

“哦，原来是这样啊，看来我们一直以来对链表太偏袒了。不过如果算上查找位置的时间，数组应该也会比链表慢吧？”

“打个比方来说，数组的插入删除就像是在做搬运工，简单而烦琐，而链表的插入删除就像是拉抽屉，每个抽屉里都有一个能够打开另外一个抽屉的钥匙。”

“对啊，所以说来开抽屉找东西要比来来回回搬运东西简单多了呢。”

“往往很多人认识到了查找位置这个操作不可忽略，但是却由于有你这种想法仍然对谬论深信不疑。你之所以认为拉抽屉比搬东西简单，就是你在其中加上了重量的因素，总是认为搬运的东西很沉，而来开抽屉省力。”

“是吗？听你这么一说好像真有点这个意思。”

“但是计算机中的数据是没有重量的，计算机关心的只是速度，对于数组，进行相邻内存内容的复制是非常快速的，而不断地进行寻址操作才是最耗费时间的。”

## 2. 谬论的证明

口说无凭，本书绝对不是“学院”派，下面就给各位读者一个证明。采用 Java 集合框架中的 ArrayList 和 LinkedList 来作为数组和链表的代表，编写一个测试 ArrayList 和 LinkedList 随机访问元素的耗时比较程序，其代码如下所示。

```

1  package wyf;
2  import java.util.*;                //引入相关包
3  class RandomAccessElementArrayListFasterThanLinkedList{
4      public static void main(String args[]){
5          long[] timeBegin=new long[2];    //记录开始时间的数组
6          long[] timeEnd=new long[2];      //记录结束时间的数组
7          Integer[] ia=new Integer[5000];  //创建一个 Integer 对象的数组
8          for(int i=0;i<5000;i++){
9              ia[i]=i;                    //为 Integer 对象数组的每个元素复制
10         }
11         int tempi;
12         Random r=new Random();           //随机数，用于产生随机访问位置
13         //对 ArrayList 进行访问
14         List list=new ArrayList(Arrays.asList(ia)); //创建 ArrayList 对象
15         timeBegin[0]=System.currentTimeMillis();
16         for(int i=0;i<100000;i++){
17             tempi=(Integer)list.get(r.nextInt(5000)); //随机进行 100000 次访问
18         }
19         timeEnd[0]=System.currentTimeMillis();
20         //对 LinkedList 进行访问
21         list=new LinkedList(Arrays.asList(ia)); //创建 LinkedList 数组
22         timeBegin[1]=System.currentTimeMillis();
23         for(int i=0;i<100000;i++){
24             tempi=(Integer)list.get(r.nextInt(5000)); //随机进行 100000 次访问
25         }
26         timeEnd[1]=System.currentTimeMillis();
27         //打印输出比较结果
28     }
29 }

```

**提示** 由于篇幅有限，打印输出代码未列出。

上述代码分别对两个长度均为 5000 的 ArrayList 和 LinkedList 进行 100000 次随机访问，打印输出窗口如图 10-3 所示。

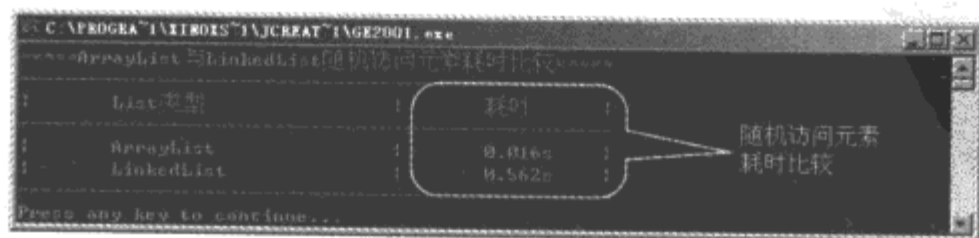


图 10-3 链表和数组随机访问元素耗时性能比较

由图 10-3 可以明确看出数组在随机访问执行上比链表快了将近 40 倍，读者也可以参考代码



自己测试一下。为了使测试更加全面，本书还对数组和链表随机增删元素的耗时比较（见图 10-4）和按顺序添加元素的耗时比较（见图 10-5）做了测试，由于篇幅关系代码未列出，读者可以在前面代码的基础上自行改动测试。

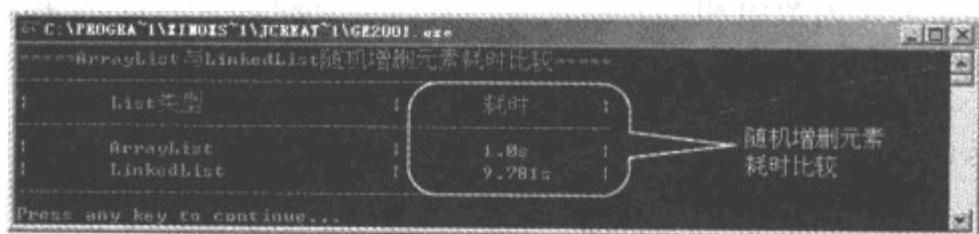


图 10-4 链表和数组随机增删元素耗时比较

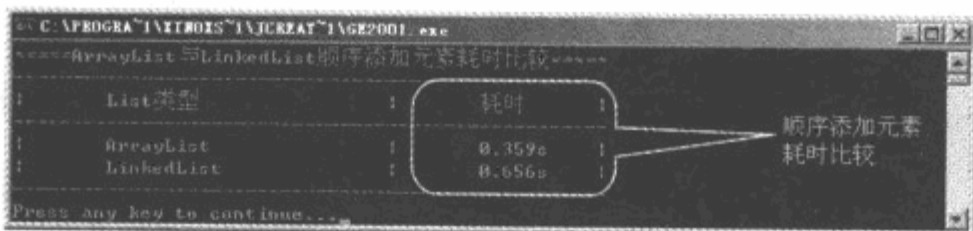


图 10-5 链表和数组按顺序添加元素耗时比较

“流言止于智者”，如此钢铁般的事实摆在眼前，相信所有人都不会再对本小节前面提到的那个谬论执迷不悟了。这个谬论听过忘了倒还算幸运，最可怕的就是像蔡佳娃那样用心的人。所以在实际开发过程中，没有特别的要求，还是要选用数组类存储作为存储数据的方式。

### 10.1.2 Java 真的比 C/C++ 慢吗

前面小节介绍的数组和链表速度差异的谬论从“古”至今一直存在，但有些结论一开始或许是正确的，但是当条件改变了以后，这些结论就可能变成了谬论。比如每个 Java 开发人员都曾经怀疑过的 Java 语言的性能问题。

“师兄，果然如你所说，我回去把 LinkedList 改成 ArrayList 之后，已经不是特别卡了，我抽空再去找找其他原因。顺便问一下，是不是 Java 语言的性能并不是很高啊？”

“怎么会这么问呢？你可不要因为你的小游戏运行不流畅就埋怨 Java 虚拟机的速度慢啊。”

“没有没有，我只是这么觉得，你看游戏是最要求性能的吧，可是市面上哪有 Java 开发的游戏啊，倒是 Java 经常运行在功能强大的服务器上面。”

“哦，你是觉得 Java 沾了大型服务器硬件好的光而其实本身性能很差喽？这种话从你嘴里说出来，我很寒心啊。”

任何一门编程语言的问世，性能问题都是开发者和用户最关心的地方，Java 也不例外。作为第一个完全面向对象的编程语言，Java 从诞生之日起，有关它和 C/C++ 之间执行效率的比较就没有停止过。较为广泛的一个认识就是，尽管 Java 语言最流行、最强大、最吸金，但是在性能问题上，仍然不是 C/C++ 的对手。

在决定孰好孰坏之前，有必要介绍一下 Java 程序的执行机制。众所周知，Java 程序运行在 JVM（Java Virtual Machine）上，单单这一点，就能让许多人对 Java 的执行效率有所怀疑了。一个 Java 程序要想被执行首先被编译成为一个 class 文件，这个类文件实际上就是一组字节命令码，由凌驾于操作系统之上的 Java 虚拟机负责对这些字节码进行解释执行。

与 Java 的 JVM 机制不同, C/C++ 通过编译器直接编译为操作系统的本地代码, 执行起来效率自然要高一些。在 Java 刚刚问世的时候, 由于各方面还不够成熟, 很多地方确实都比不上老一辈的 C/C++。

对于 Java 程序的运行效率问题, 笔者深有感触, 笔者于 1998 年开始从事 Java 开发, 那时的 JDK 版本是 1.0 或者 1.1。当时用 Java 为国家重点实验室写了一个比较大的 CORBA 程序, 调试了一个多月, 总是出现同一个问题: 一运行就死机, 一死机就要手动重启服务器。

有一次死机之后需要重启, 正赶上午饭时间, 心想吃完饭回来再重启吧。结果吃饭回来后发现程序已经运行出结果了, 原来不是程序有问题, 而是 Java 运行效率的问题! 这让当时的笔者对 Java 之路的光明前景产生了迷茫与怀疑, 着实让笔者郁闷了一段时间。

现在的 Java 执行效率还是这么慢吗? 随着 Sun 等大公司对于 Java 语言的不断优化, Java 的性能也在不断改善。2004 年 9 月推出的 Java SE 5 版本是 Java 发展的重要里程碑, Java SE 5 除了在性能上提升了 Java 执行效率之外, 还推出了一个重要的特性: 热点技术和 JIT (即时编译) 技术。

在一个服务器级的程序中, 必然要频繁接受客户端的访问。如果某一个局部的程序模块被多次访问, 而其执行效率又相对较慢, 则这个模块就成为整个系统的瓶颈。这些访问频率高、严重制约系统性能模块就称为热点 (Hot Spot)。

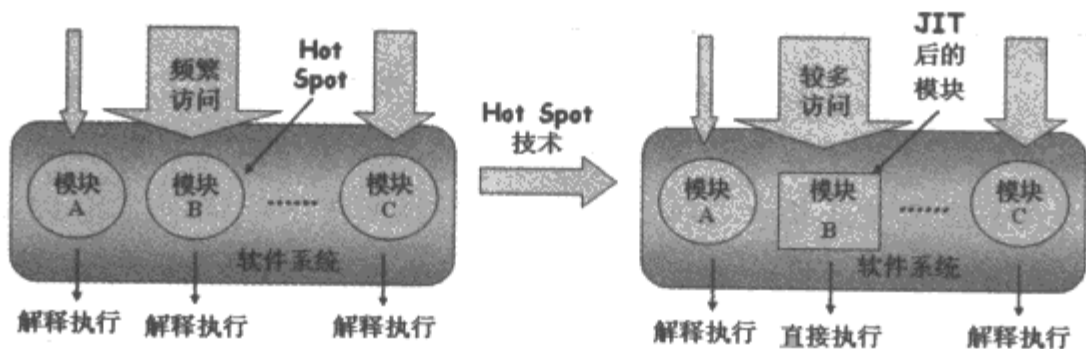


图 10-6 Hotspot 技术示意图

对于系统中的热点, Java 虚拟机中会用 JIT (Just In Time) 技术, 将系统热点的字节码翻译成操作系统的本地代码。这样以后再有用户调用这个模块的时候, 虚拟机不会再将其解释执行, 而是直接调用其本地代码。其原理如图 10-6 所示, 这样执行起来就与 C/C++ 是一样的工作方式了。

Java 虚拟机会自动检测系统的瓶颈, 然后将其转变为操作系统本地代码并进行调优, 这种模块执行的自动优化是 C/C++ 所不具备的。

目前 Java 的最高版本是 Java SE 6, 虽然 Java SE 6 并没有推出能和 Hot Spot 媲美的新特性, 但是 Java SE 6 在极大程度上优化了自身, 大大提高了程序执行速度。从现在来看, 解释执行的 Java 已经不比编译执行的 C/C++ 慢多少了。

光说不练没有用, 按道理还是要拿出证据来才行, 历史上 C++ 和 Java 是做过真实 PK 的。PK 的结果可以用图 10-7 来表示, PK 是在 LINPACK 基准和 SciMark 基准上进行的。

在这里有必要简单介绍一下 LINPACK 基准和 SciMark 基准, 这两种基准都是采用数学计算的方法测试软硬件系统的性能。LINPACK 全称是 Linear system package, 即线性系统软件包, LINPACK 的测试内容是将一元  $N$  次稠密线性代数方程组用高斯消元法求解; 而 SciMark 的测试内容相对丰富些, 有快速傅里叶变换、稀疏矩阵乘法、连续松弛迭代等测试要素。

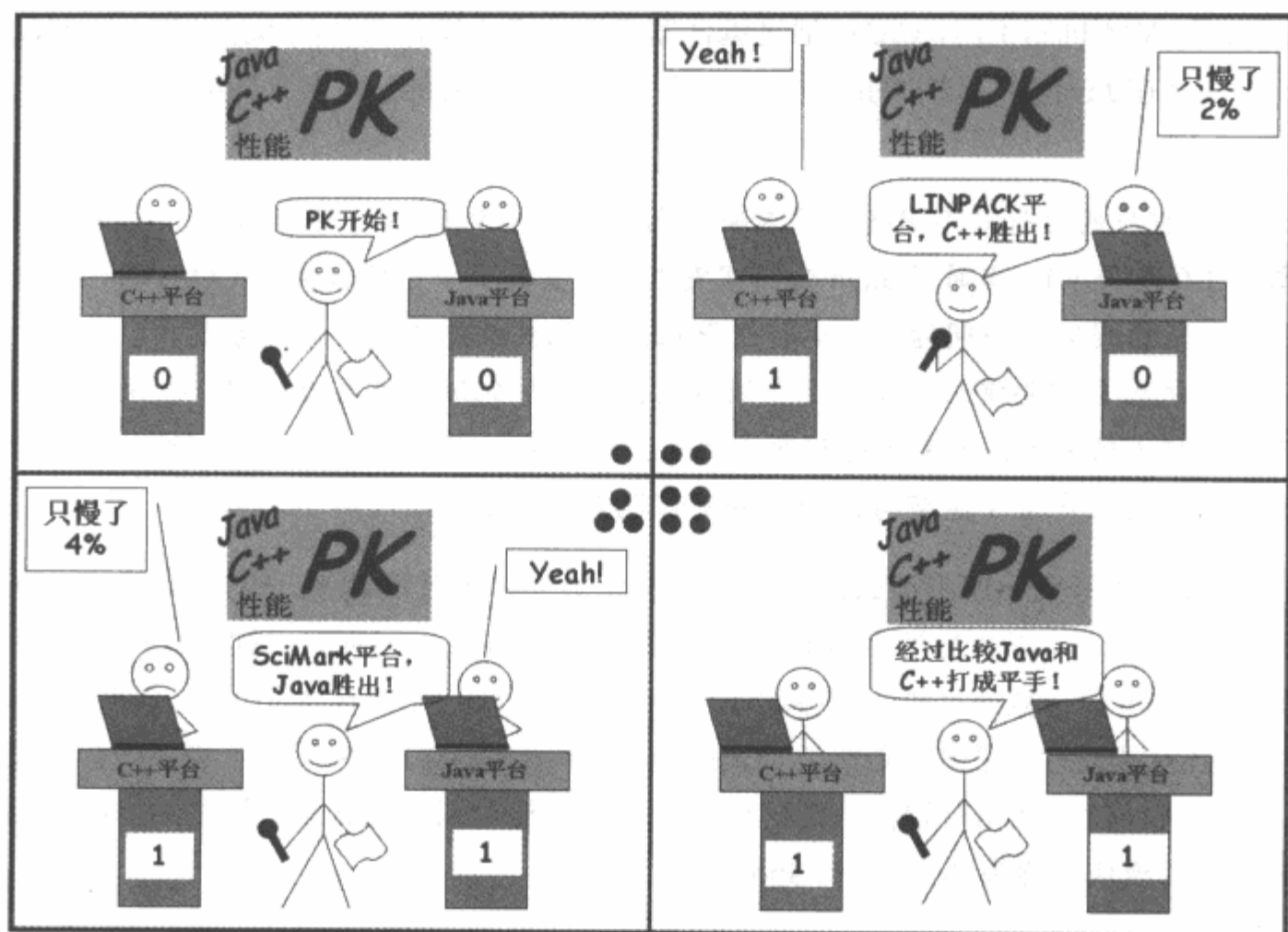


图 10-7 C++和Java性能PK

这两种测试基准被广泛用于各类计算机系统性能的测试，特别是在大型机、巨型机方面。比如前几天我国刚刚在天津国家计算中心研制成功的巨型机天河一号就是使用 LINPACK 基准进行实际性能测试的。天河一号的研制成功大大提升了我国的大型计算的能力，笔者作为一名 IT 人也非常高兴。

由图 10-7 可以看出，当下 Java 和 C++ 二者之间的优势和劣势之差很小。C++ 在 LINPACK 平台上性能比 Java 快 2%，而 Java 在 SciMark 平台上也只优于 C++ 4%，可以说 Java 和 C++ 之间在性能方面已经基本不相伯仲。

同时，要说运行快，高级语言中谁也快不过 Fortran，这个为科学计算而生的语言是编程界每个语言都乐于对比和渴望超过的“偶像”。Java 和 Fortran 相比，在进行矩阵运算时大部分情况速度接近，这已经是个让其他语言非常眼红的成绩了。

**提示** 如果说现在再遇到笔者十几年前遇到的“死机式”的程序运行效率，不要再怀疑 Java 的虚拟机了，一定是自己的程序有着不够完善的地方，需要闭关修炼喽。

## 10.2 迷信工具，缺乏纯代码能力

如今的 Java 开发人员在工作时，必然不会在简陋的记事本或 VI 中挥汗如雨，手边肯定会有数不清的工具保驾护航。工具就是用来辅助开发，起到润滑剂的作用，真正的发动机还是开发人员的脑子。如果过分迷信这些开发工具，将其作为工作和学习的重点，一旦碰到赤手空拳的场合，荒废已久的武功可就再也顶不上大用处了。

### 10.2.1 迷信 ORM

ORM (Object/Relational Mapping) 前面的章节已经有所介绍, ORM 通过将面向对象编程的开发人员所不擅长的关系型数据库处理包装成对象, 使得关于数据库的操作对开发人员彻底透明化, 在很大程度上简化了开发。

不过就像视频格式转换也会有失真, ORM 也做不到关系到对象无代价的完美转换。生产环境中有些情况使用 ORM 做的效果并不够好, 或者有些问题干脆就没办法用 ORM 来做。如果懒惰的开发人员过分地迷信 ORM 的话, 会很难让自己有技术水平上的真正提高。

“师兄啊, 我好恨你啊!”

“啊? 这是为什么啊? 你确信是在跟我说吗?”

“师兄, 你上次对我说 ORM 对我们 OO 开发人员帮助很大, 我回去之后在工作中就频频使用, 一开始觉得还可以。但是后来我发现有的时候 ORM 的效率也没有那么快嘛, 害得我还得去请教我们公司的数据库分析师, 最后用 SQL 搞定。”

“哎, 你怎么这么傻呢。关系数据库系统发展到现在也有三四十年了, 哪能让你用个 ORM 工具就完全替代啊? 对于 ORM 这类快捷方式的工具, 会用是必需的, 但是千万不可迷信哪。”

ORM 并不是不好, ORM 的思想是很不错的, 只是不能将大小事务全权委托于 ORM, 因为 ORM 也有做不了和做不好的事。

#### 1. 做不了的事

在面对一些复杂的检索时, 尤其是需要进行多层嵌套的统计类检索, ORM 就显得力不从心了。而这些复杂的问题, 采用 SQL 的高级语法是可以轻松完成的。

在 ORM 里, 主要的任务就是将数据映射成对象, 由此产生的对象与对象之间的关系要么是一对一关系, 要么是一对多关系, 或是多对多关系, 这目前是 ORM 的极限了。而在实际生产环境中有很多情况都需要进行各种复杂的统计操作, 并不是要某个具体对象的信息, 这些工作对 ORM 来说就显得有点力不从心了。

“蔡佳娃, 我们来举个例子, 假设你需要对一家超市的商品数据库进行检索, 找出这家超市的特定部门如熟食部都卖什么, 你应该怎么做?”

“很好办啊, 用 ORM 会很简单。”

“的确, 这是典型的一对多关系, 一个部门对应多种商品, 这种问题用 ORM 工具来解决十分高效。然后你这样想, 超市和商品的制造商是合作关系, 如果我要问你这家超市最有价值的合作伙伴是什么, 你应该怎么做呢?”

“呃, 我想想啊, 应该先找最赚钱的商品, 再去找商品的生产厂商……”

“考虑欠佳了吧! 要知道有的厂商并不是只有一种产品的啊, 比如纳爱斯这个公司既有洗衣粉和肥皂还有牙膏, 就算牙膏卖得不好, 也要算到纳爱斯这个公司里面的呀。”

故事中的这个问题如果用 ORM 工具来做的话将会变得很复杂, 因为这并不是一个简单的对象和对象的对应关系, 为了找出超市最有价值的合作伙伴, 有可能需要将上万种商品销售信息提取生成对象来进行统计。这样的工程是很不实际的, 所以目前对于复杂的统计类问题主要还是靠



SQL 来解决的。

## 2. 做不好的事

有些问题一般情况下可以使用 ORM 来解决，但是在某些 ORM 系统中，有时执行效率将会大大降低，产生诸如“1+n”之类的问题。“1+n”问题就是指如果用一句 SQL 就解决问题了，但是经过 ORM 转化后反倒变成了  $n+1$  句 SQL 语句。

比如查询名叫 Tom 的车主名下所有的车，正常的 ORM 系统第一步先把 Tom 作为对象加载，第二步检索 Tom 名下的车的相关信息，所以在 ORM 中这个过程将会被转化为两句 SQL，一句查询 Tom，一句查找 Tom 名下的车。

但是在有些情况下，有些 ORM 系统加载 Tom 对象只用了一句 SQL，查找车的信息时却变成了  $n+1$  句（ $n$  为车的辆数）。这  $n+1$  句中第一句是查 Tom 名下所有车的 ID，并将其存放在一个集合里，然后再对集合中每一个 ID 执行一句 SQL 检索，查找车的具体信息，如图 10-8 所示。

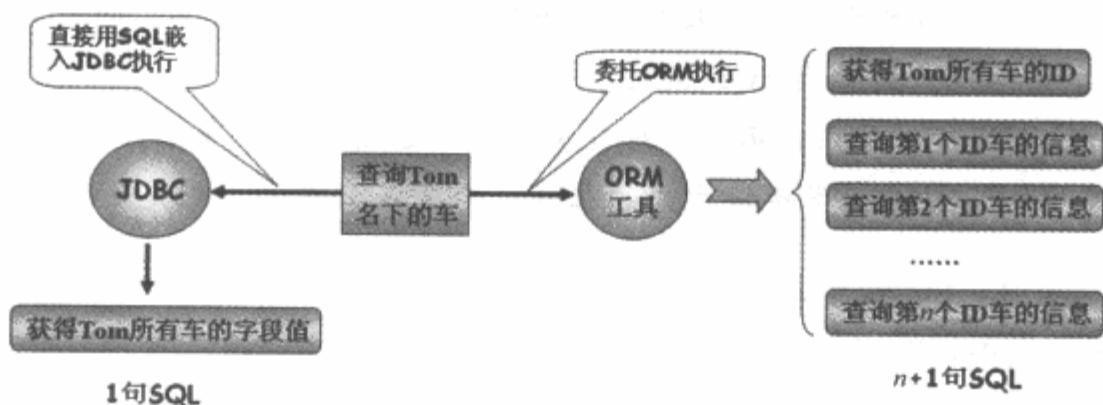


图 10-8 ORM 的  $n+1$  执行缺陷

这种  $n+1$  问题在规模不算大的情况下，或许还可以忍受。但是如果  $n$  值变大，比如说为 1000 的话，那么执行效率将会降低很多。而对于服务器应用来说，这种性能的降低是致命的。这种 ORM 使用不当的后果，绝对不是开发人员希望看到的。

除了以上两个方面，ORM 在实际应用中还存在着一些不足的地方，如下所列。

- 无法和 DBA 有效协作，开发人员并不关心数据库操作，结果 ORM 产生的 SQL 语句造成系统变慢。而由于是自动产生的 SQL 语句，数据库分析师 (DBA) 也无法改动，这样 ORM 产生的低效语句就变成了双方都无法触及的两不管地带。
- 产生大量垃圾数据，ORM 封装对象的时候必然会将数据库中对本次检索无用的字段也封装进来，这无疑加大了系统负担，将性能主要浪费在对象的创建上。
- 有的情况下需要操作的数据无法对象化，比如说只是对某个范围内的自动编号进行操作等，这时 ORM 就没有用武之地了。
- 对大二进制字段和备注字段还不能用 ORM 自动处理，需要开发人员自己结合 SQL 开发解决。

ORM 是一种出色的工具，能帮助开发人员解决一些简单、重复的数据库操作问题，但是并不能面面俱到，ORM 有时是牺牲系统的执行效率来实现面向对象的数据操作的。如果强行让其处理比较复杂的数据库任务，系统将会把大量时间花费在 Java 与 SQL 交互之间而降低总体的执行效率。

因此身为开发人员，还是不要过分迷信 ORM 带来的便捷，应该使用 ORM 的地方可以恰当使用以简化开发，提高效率。在有特定需要时应该有白手起家、用 SQL 直接开发的能力才行。



## 10.2.2 神化 IDE

IDE 即集成开发环境 (Integrated Development Environment), 对于开发人员来说, IDE 就是战场上最可靠的武器。使用 IDE 工具可以使开发人员免去非核心代码的烦琐开发, 能够更好地集中精力研究核心业务, 从而提高生产效率。不过有些开发人员对待 IDE 的方式却是错误的, 本节将会对这方面的问题进行一些探讨。

辅助 Java 开发人员的 IDE 也有很多, 这些工具几乎都是开源免费的, 本小节将主要介绍一下应用最为广泛的 NetBeans 和 Eclipse。

### 1. NetBeans 开发工具

NetBeans 的开发界面如图 10-9 所示, 它是 Sun 为广大开发者奉上的一款全功能的 Java IDE。NetBeans 针对目前几乎所有的操作系统平台都有相应的版本, 在 NetBeans 上可以进行几乎所有 Java 程序的开发, 从 EJB 到 Java ME 的 MIDlet, 从 Swing 界面到 JSF 界面, 同时 NetBeans 中还自带了 Tomcat 和 GlassFish 服务器, 免去了另外安装的麻烦。NetBeans 也支持像 VB 程序开发那样的控件拖拉、所见即所得的方式。

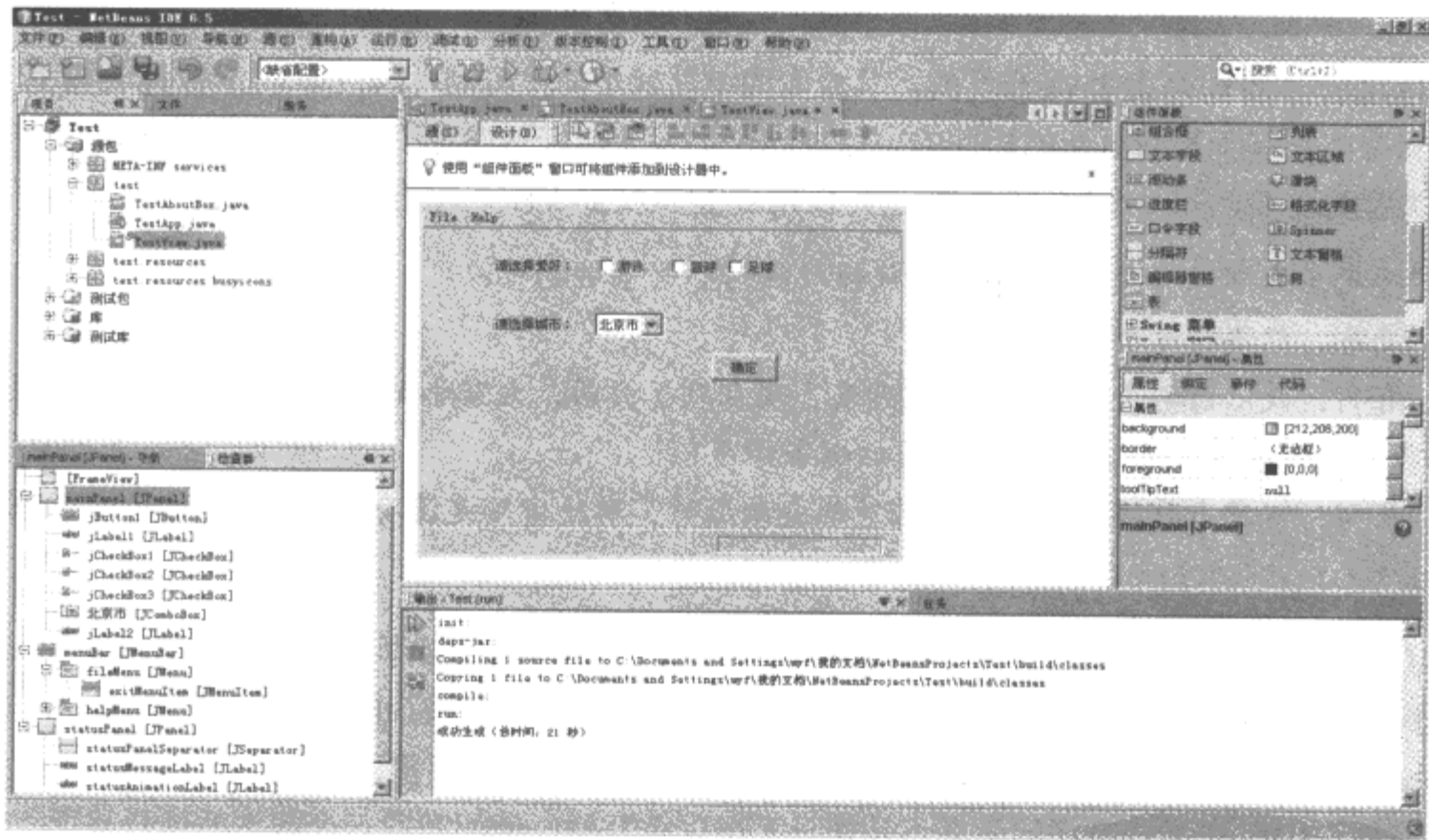


图 10-9 NetBeans 开发界面

**提示** 通过安装一些插件, NetBeans 还可以进行 C++ 等其他语言的开发, 这在各种开发工具中也是不可多得的一款功能。

### 2. Eclipse 开发工具

Eclipse 的开发界面如图 10-10 所示, Eclipse 是最受广大 Java 开发者青睐的 IDE。它原本是 IBM 的一个项目, 2001 年 IBM 将 Eclipse 捐献给开源社区, 目前 Eclipse 的最高版本是代号为 Galileo 的 3.5 版。

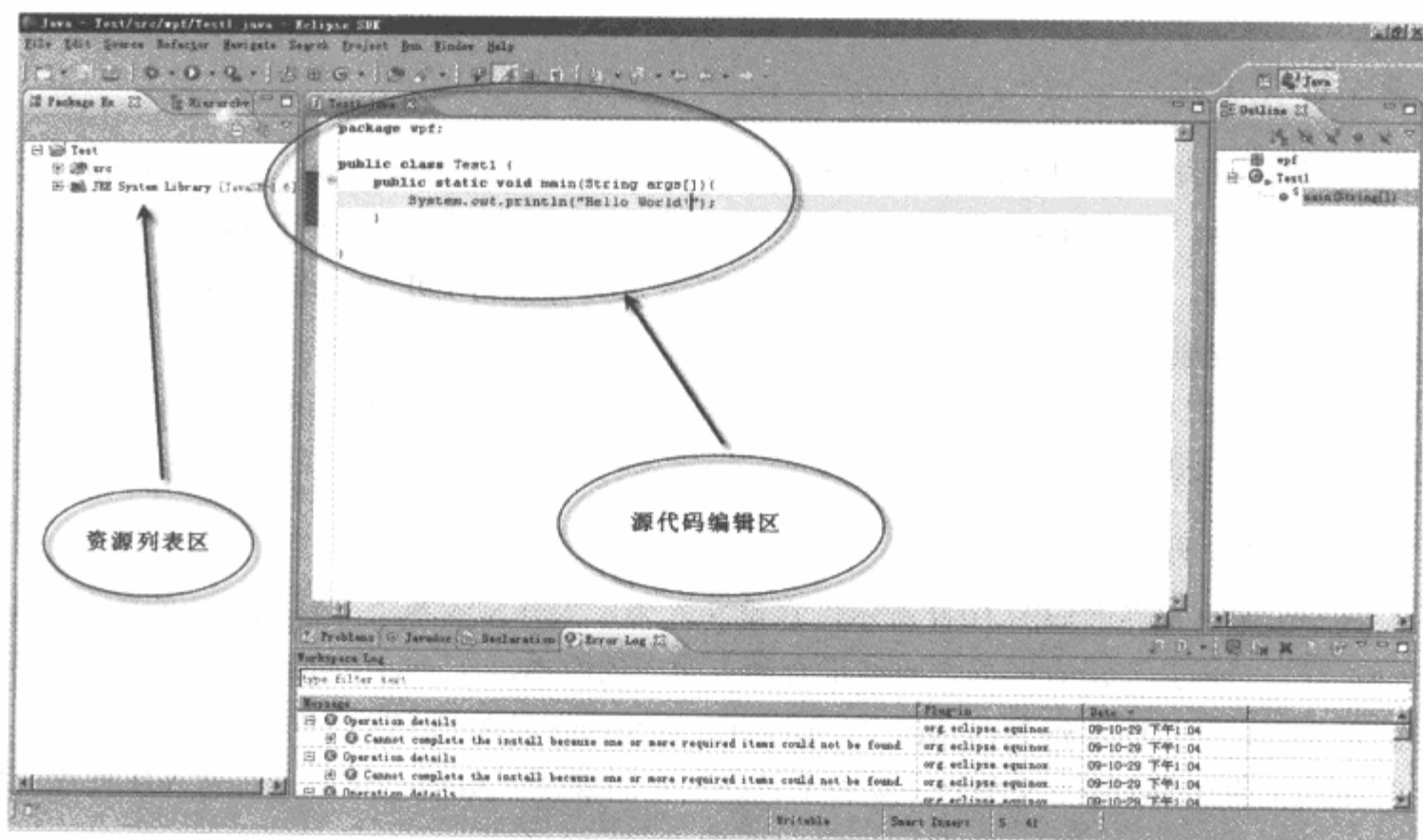


图 10-10 Eclipse 开发界面

Eclipse 本身即是由 Java 开发的，其最大的特性就是插件机制，Eclipse 对于各种 Java 程序开发的支持都是基于不断更新的插件的。这种高可扩展性很好地满足了开发人员需要不断改变开发项目类型的需要，同时在插件的作用下，Eclipse 也支持如 VB 一样的可视化编程。

“师兄，你那里有没有 Eclipse 的视频教程啊？电子书也可以。”

“没有，你要这个有用吗？”

“有啊，我以前都是用 Jcreator 做开发，现在到了公司，大家都是 Eclipse，就我的基础差些。我决定了，要对 Eclipse 进行惨无人道的刻苦学习！”

“呵呵，不过我要提醒你，不要捡芝麻丢西瓜呀，职场中可是没有 Eclipse 使用师这个职位的哦。”

身为 Java 开发人员，应该深入学习 Java 技术。但是很多人随着学习和工作的不断深入，渐渐体会到 IDE 给开发带来的便利，开始进行工作重心的转移，把精通 Java 的战略目标改为了精通 NetBeans、Eclipse 等功能强大的 IDE，这就有些买椟还珠、本末倒置了。

诚然，IDE 在某些方面的效率不是两只手可以企及的，例如开发一个静态的用户界面，只需要将控件进行来回拖拉即可完成。再例如对数据进行封装，只需要一个命令，就可以为开发人员省去烦琐且毫无技术含量的 get 和 set 访问器的编写。

但是，说到底 IDE 也不过是机器，做的都是些死工作。客户项目中最有价值的是活的东西，不然所有人只要精通 Eclipse 即可无须再关注 Java 了。当需要满足客户这种动态的需求时，就只有靠开发人员的创造性能力了。

“蔡佳娃，我们来做一个简单的 PK，你就明白为什么不能全靠 IDE 做事了。一个高度依赖 IDE 的开发人员虽然在一些固定模式的工作中会胜他人一筹，但是碰到需要动态变化的地方 IDE 就不灵光了，这个时候靠的就是纯代码的功力了。”（见图 10-11）

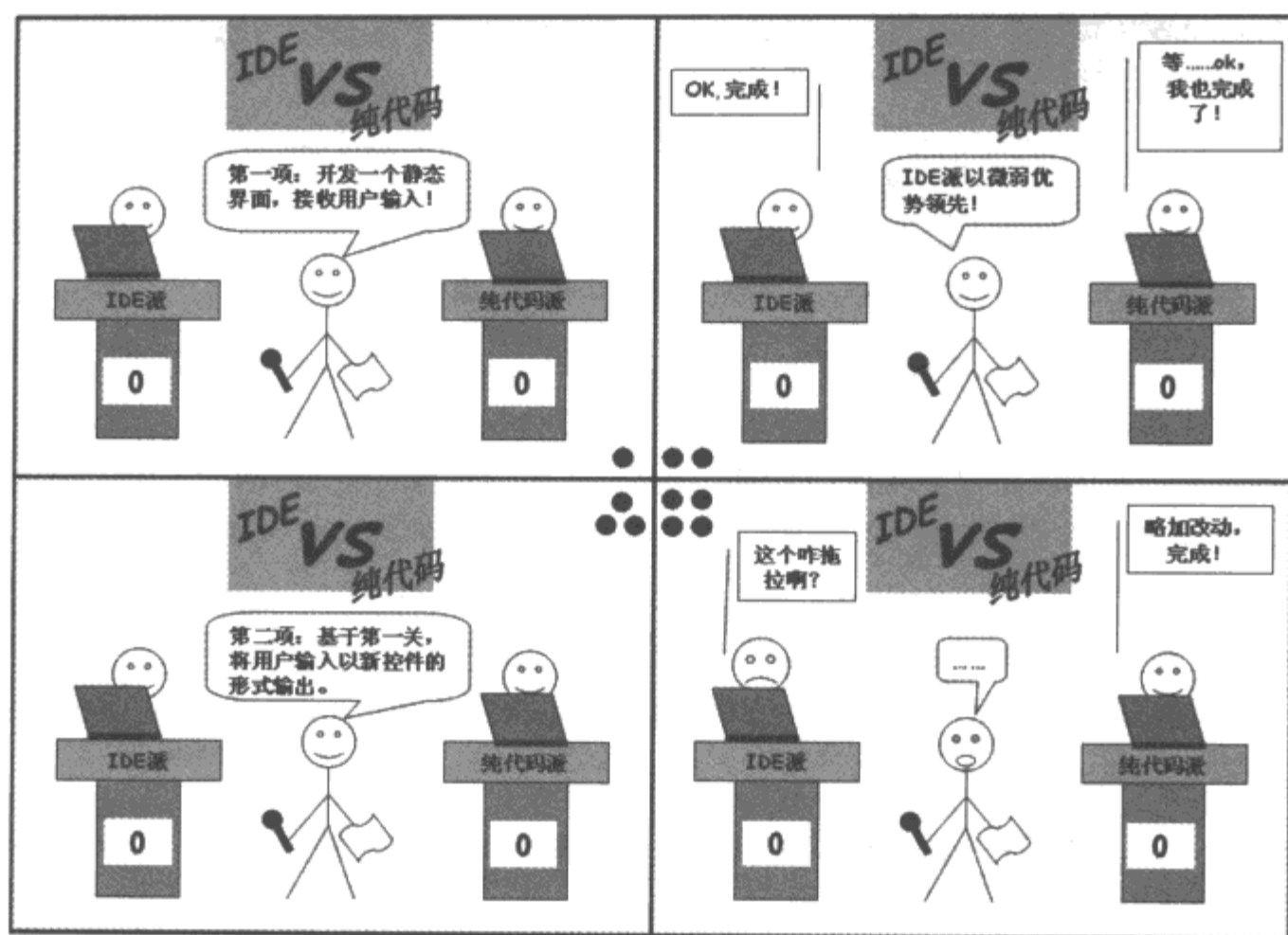


图 10-11 IDE 派和纯代码派的 PK

“师兄，你具体点说呗。”

“好，比如说你需要开发一个用户注册模块的界面，这种情况下用拖拉的确很方便，因为所有的模块都是静态的，不管谁来写都是这样的形式。”（如图 10-12 所示）

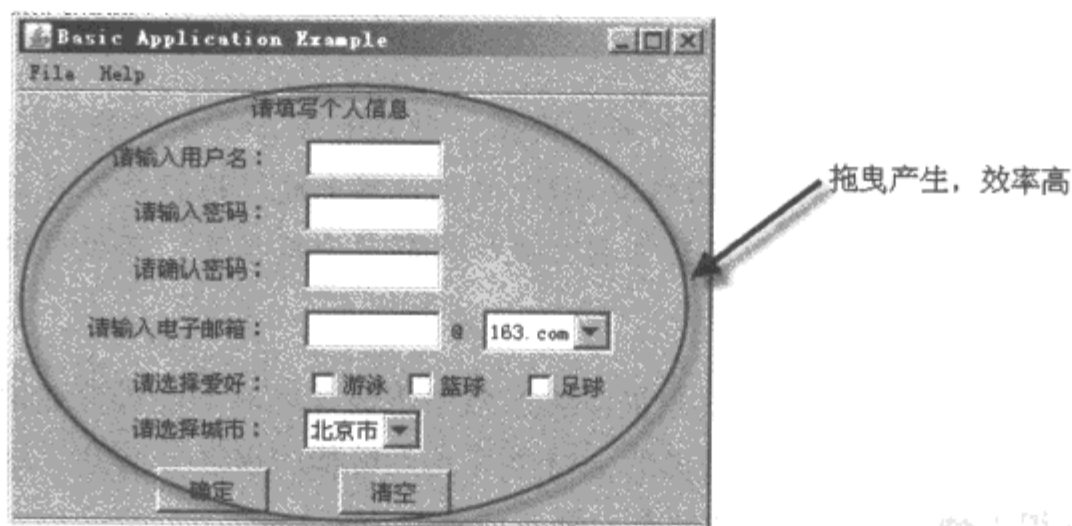


图 10-12 使用 IDE 拖曳产生程序界面

“对啊，用 IDE 的设计界面进行拖拉定位比一次一次地试代码要方便快捷得多呢。”

“可是如果我给你的问题是：根据用户的输入来动态生成新控件并添加到程序中，这时候你该怎么办呢？拖拉就不行了吧？”

“拖拉都是在编译之前一次性做好的，而用户的输入是动态产生的，控件也都是动态产生的，这个用拖拉显然是搞不定的嘛。”

“所以说有的时候一些工作 IDE 也是鞭长莫及的，这个题目如果自己用手工写的话虽然控件布局开发方面要慢一些，不过这个动态功能的实现肯定是没有问题的。”（如图 10-13 所示）

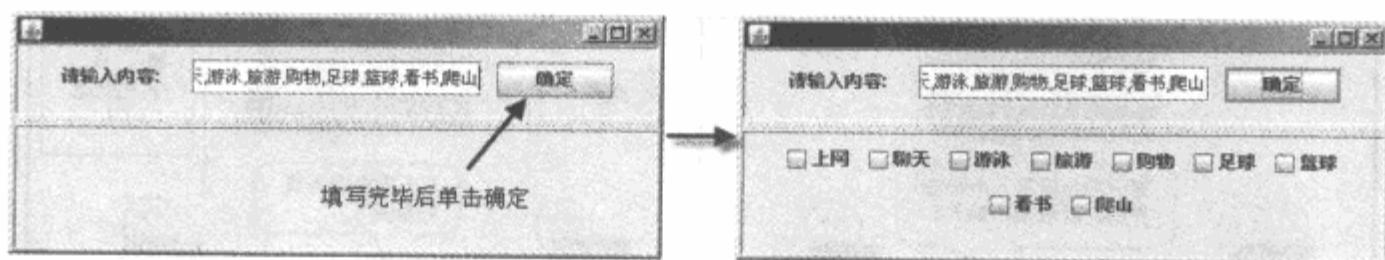


图 10-13 使用纯代码编写的动态桌面程序

用 Java 写一些动态产生控件的代码，对于一般开发人员来说都不算难，只需要在某个事件处理方法里面新建控件对象并添加到容器中即可。但是对于 VB、Delphi 的开发人员来说，动态产生控件就很困难了。原因就是很多 VB、Delphi 的开发人员错误地将学习 IDE 作为了工作重点，对于开发工具的依赖已经大大超过手工代码，所以超出拖拉范围的内容就做不了了。

但是 Java 开发不比 VB、Delphi，完全依赖鼠标的拖拉是不行的。Java 开发面向的都是大型应用，动态性要求很高。IDE 工具帮助我们完成了一些固定模式下的“死”工作，很大程度上提高了开发效率，这是开发人员使用 IDE 的主要动机。但是很多创造性的工作并非是看似强大的 IDE 工具所能够包干完成的。

“蔡佳娃，这还只是 IDE 的一个不足，我们下面再来谈谈 IDE 工具另一个不尽如人意的地方。”

“哦，这次是什么啊？”

“不管是进行界面编程还是 EJB 开发，你在前台拖拉或是点来点去，而 IDE 工具在后台基于一定的规则自动帮你生成源代码，这是 IDE 拖曳开发的基本工作机制。”

“嗯，是啊，说到底 IDE 工具也就是帮我们写了点源代码。”

“如果仅是一些代码的框架，是可以放手交给 IDE 工具自动生成的，例如设计器自动生成一个只含有参数类型和返回值的空方法，由开发人员去实现方法体，这种情况下还是可以信任 IDE 的。”

“是啊，像一些界面编程中的控件大小布局问题，用 IDE 不知要方便多少倍呢。”

“但是如果说一些复杂的业务逻辑仍然由 IDE 的设计器来通过拖拉等方式自动生成的话，虽然原则上不会错，但是需要注意的是 IDE 工具生成的代码冗余度很高，不该考虑的细节也算进去了，效率也就相对较差了。”

真正的高手应该通过恰当使用 IDE 来提高生产效率，但是绝对不能过分迷信 IDE。什么能让 IDE 去做，什么是开发人员必须亲手去做的，每个开发人员都应该区分清楚。这样就能在不影响软件产品质量的前提下最大限度地提高效率，降低成本。

## 10.3 浅尝辄止，孤陋寡闻

Java 技术发展到今天，虽然历史不够悠久，但也算得上是博大精深了。在广而深的 Java 技术海洋中，很多技术对于 Java 开发人员来说或者对其研究得不深，浅尝辄止，又或者很少听过，有些孤陋寡闻。本节就简单列举几个入门级开发人员重视不够和了解不多的技术。

### 10.3.1 finally 的忽视

繁忙的工作之余，也不管人家乐不乐意，蔡佳娃总是喜欢跑到师兄牛开复的家里逗留，这不

又在师兄的电脑上倒腾起来。

“噢，师兄，你桌面上的 stain 文件夹是干什么的？stain 不是污点的意思吗？我能打开来看看不？”

“OK，打开吧，里面没有什么隐私性的东西，主要是我以前学习 Java 的时候记下来的一些疏忽或是了解太少的知识，我称这些为‘污点’，你要是觉得有用可以看看。”

“师兄的手稿当然有用了，我要赶紧拜读一下……噢，finally 语句的忽视，这个只有一行，具体讲的是啥啊，师兄？”

finally 想必每位读者都知道，这是用来修饰语句块的关键字，通常和 try/catch 语句联合使用。当 try 语句块代码在执行的过程中因抛出异常而退出程序之前，要先执行 finally 语句块。finally 语句虽然很简单，但却是非常有意义的，而这也是很多菜鸟都会忽略的地方。

本小节把对 finally 语句的忽视分为两种情况，将以一个基于 Socket 协议的客户端与服务端进行数据传递的代码片段为例来讨论 finally 的重要性。

### 1. 根本就不需要

有些菜鸟对于 finally 根本就不重视，也不会在开发中去使用，这种类型的菜鸟开发的 Socket 客户端和服务端的交互代码如下所示。

```

1  try{
2      Socket sock = new Socket("127.0.0.1",9999);           //创建 Socket 对象
3      DataInputStream din = new DataInputStream(sock.getInputStream());
                                                                //打开输入流
4      DataOutputStream dout = new DataOutputStream(sock.getOutputStream());
                                                                //打开输出流
5      //业务代码
6      .....
7      din.close();                                           //关闭输入流
8      dout.close();                                           //关闭输出流
9      sock.close();                                           //关闭 Socket 连接
10 }
11 catch(Exception e){
12     //异常处理程序
13 }
```

这应该是所有开发人员都深恶痛绝的一种做法，虽然能够记得将创建的输入输出流及套接字本身关闭，但是却忽略了在关闭上述对象时可能产生异常的情况。因为一旦出现异常，将不会执行后面的代码，这样无用的对象将会占用宝贵的内存。

finally 这种保证某段代码一定执行的特性在开发大型应用的时候非常有必要。对于有些代码，不管执行过程中会不会抛出异常，都有一些工作是必须要做的，比如与数据库的连接过程中，关闭连接就是必须要做的一件事。使用 finally 语句可以解决这类问题，从而提高系统的安全性和可靠性。

### 2. 用了没用好

有很多人都会意识到上面提到的要点，所以在编写代码的时候，会在 try/catch 语句之后加上



finally 语句以保证释放某些资源。且看如下的代码。

```

1   Socket sock = null;                                //声明 Socket
2   DataInputStream din = null;                        //声明数据输入流
3   DataOutputStream dout = null;                     //声明数据输出流
4   try{
5       sock = new Socket("127.0.0.1",9999);          //创建 Socket 连接
6       din = new DataInputStream(sock.getInputStream()); //打开输入流
7       dout = new DataOutputStream(sock.getOutputStream()); //打开输出流
8       //业务代码
9       .....
10      din.close();                                    //关闭输入流
11      dout.close();                                   //关闭输出流
12      sock.close();                                   //关闭 Socket 连接
13  }
14  catch(Exception e){
15      //异常处理程序
16  }
17  finally{
18      try{
19          din.close();                                //在 finally 语句块中关闭输入流
20          dout.close();                               //在 finally 语句块中关闭输出流
21          sock.close();                               //在 finally 语句块中关闭 Socket 连接
22      }
23      catch(Exception e){
24          //异常处理程序
25      }
26  }
```

乍看这段代码充分考虑了可能出现的异常，已经比较完善了。但是仅仅做到这里还不够，和第一种人相比只能算是“五十步笑百步”，原因就是并没有将这种“保证执行”做到位。

如果上述代码在执行到第 19 行的时候不幸抛出了异常，那么第 20 和 21 行的对象关闭操作将不会执行，这样仍然不能满足要求。真正在开发过程中，应该将上述代码中的 finally 语句块改为如下形式。

```

1   finally{
2       try{ din.close(); }                             //关闭输入流
3       catch(Exception e){
4           //异常处理程序
5       }
6       try{ dout.close(); }                             //关闭输出流
7       catch(Exception e){
8           //异常处理程序
9       }
10      try{ sock.close(); }                             //关闭 Socket 连接
11      catch(Exception e){
12          //异常处理程序
13  }
```

```
14 }
```

这样虽然代码比以前复杂许多，但是却真正保证了无论哪里抛出异常，都会将输入输出流关闭并释放 Socket 对象。

这里有必要再谈一谈“资源泄露”的问题，其实说白了，finally 语句的作用就是及时释放资源以不影响系统性能。在系统中申请了资源，如果这个资源是一定要释放的，那么就应该遵循“最晚申请，最早释放”的原则，就近释放。

就近释放是指在 A 类的一个方法中申请了某个资源，就应该在这个方法的末尾释放掉。如果做不到，那么至少在 A 类中开发另一个方法负责将资源释放掉，绝对不可以申请了资源却将其传给其他类的对象以期待别人将其释放，这样就很容易产生“资源泄露”。而且就算是在同一个类中，也很容易遗漏开发资源的释放方法。


举个例子来说，在开发 JSP 程序时，在某个 Java Bean 的一个方法中需要检索数据库将所有信息返回，按如下方式开发就很不恰当。

```
1 public ResultSet getXxx() {           //方法返回结果集
2     //业务代码
3     .....
4     return rs;                         //rs 为 ResultSet 型引用
5 }
```

如果按照这样的方式开发，ResultSet 在完成数据库检索之后不能关闭，必须将其引用传回 JSP 页面提取出数据后再释放掉，这样就很不安全了。ResultSet 是在 Java Bean 中申请的资源，最好就在同一个方法中释放掉，例如应该将上述代码改为如下形式。

```
1 public ArrayList<xxx> getXxx() {       //将数据提取后封装在集合框架中返回
2     //业务代码
3     .....
4     return al;                         //al 为 ArrayList<xxx>型引用
5 }
```

这样结果集就可以在 getXxx 方法中关闭，避免了依赖别人释放资源带来的风险。

 **提示** 笔者强烈建议读者朋友在开发的过程中编写申请资源的代码时就将释放资源代码完成，然后再去开发中间的业务代码。如果能将其放在同一个方法中最好，如不能至少也要有个释放的方法与之对应，绝对不可以申请资源后将引用传递给别的对象。这样的 bug 一旦在实际中遇到，是很不容易处理的。

### 10.3.2 PreparedStatement 的误解

PreparedStatement 即预编译语句，它继承自 Statement。相比于 Statement，使用 PreparedStatement 执行大业务量的数据库操作，可以大大提高 SQL 语句的执行效率。但是如果对预编译语句理解不到位，使用错误的话，其产生的后果也是非常糟糕的。

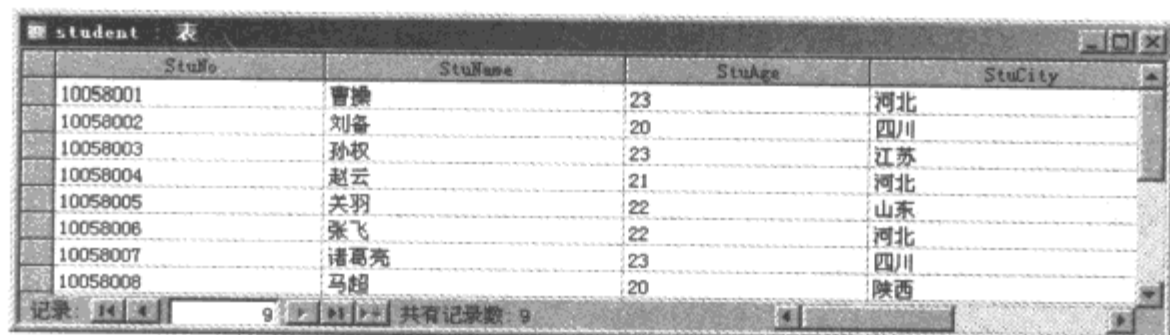
“师兄，你污点文件夹里的好多问题我都没见过，不过有些问题好像并不是特别有名啊，比方说这个 PreparedStatement 的误解问题具体指什么啊？”

“这个啊，说的是如果对于 PreparedStatement 使用得当的话，可以比 Statement 性能高许多，但是如果使用不当，系统的性能就会变得更差了。”

“啊，会这样吗？”

“我们可以来做一个对比，看看如果对 PreparedStatement 理解不深入会带来什么结果。”

本例中将分别对 Statement 和 PreparedStatement 的正确与错误使用做一个对比，假定现在需要对一个学生信息的数据库进行检索操作，希望数据库返回指定学号的学生信息。为方便对比其性能，每种检索方法进行 10000 次检索，目标数据库如图 10-14 所示。



StuNo	StuName	StuAge	StuCity
10058001	曹操	23	河北
10058002	刘备	20	四川
10058003	孙权	23	江苏
10058004	赵云	21	河北
10058005	关羽	22	山东
10058006	张飞	22	河北
10058007	诸葛亮	23	四川
10058008	马超	20	陕西

图 10-14 数据库中的 Student 表

## 1. Statement 语句的使用

Statement 语句正确用法的代码片段如下所示。

```

1  Statement st=con.createStatement();           //创建 Statement 对象
2  start=System.currentTimeMillis();             //获取起始时间
3  for(int i=0;i<10000;i++){                     //进行 10000 次对学号为 1005808 的检索
4      ResultSet rs=st.executeQuery("select * from student where stuNo='1005808'");
5      rs.close();
6  }
7  st.close();                                   //关闭 Statement 语句
8  end=System.currentTimeMillis();              //获取结束时间

```

在第 1 行创建了一个 Statement 语句对象，然后在第 4 行通过重复提交 SQL 语句执行检索。下面来看看错误 Statement 语句的用法，代码片断如下所示。

```

1  start=System.currentTimeMillis();             //获取起始时间
2  for(int i=0;i<10000;i++){
3      Statement stTemp=con.createStatement(); //创建语句对象
4      ResultSet rs=stTemp.executeQuery("select * from student where stuNo='10002'");
                                           //执行检索
5      rs.close();                             //关闭结果集
6      stTemp.close();                         //关闭语句对象
7  }
8  end=System.currentTimeMillis();             //获取结束时间

```

这种用法将语句对象的创建放到了 for 循环体中，每次进行循环的时候都会创建一个 Statement 对象，这样无疑增加了无用的系统开销。对于工作任务不是很重、效率要求不高的情况，这种写法勉强可以应付，但是如果是针对服务器级程序，这种写法就很糟糕了。

## 2. PreparedStatement 语句的使用

因为 Statement 每次都需要对 SQL 语句进行编译,相比之下,使用 PreparedStatement 处理一些大业务量和反复执行的任务就很高效率了。不过如果不正确地使用 PreparedStatement,其效率可能比错误使用 Statement 还要差很多。在本例中正确使用 PreparedStatement 的代码如下所示。

```
1 PreparedStatement ps=con.prepareStatement("select * from student where stuNo=?");           //创建语句对象
2 start=System.currentTimeMillis();                                                       //获取起始时间
3 for(int i=0;i<10000;i++){                                                                //进行 10000 次检索
4     ps.setString(1,"1005808");                                                         //设置相关参数
5     ResultSet rs=ps.executeQuery();                                                     //执行检索
6     rs.close();                                                                           //关闭结果集
7 }
8 ps.close();                                                                               //关闭 PreparedStatement 语句
9 end=System.currentTimeMillis();                                                         //获取结束时间
```

上述代码中的 PreparedStatement 对象创建在循环检索之前,由于 PreparedStatement 已经预编译了,每次只是传递参数进行调用而已,所以执行速度非常快。下面来看一看错误使用 PreparedStatement 的代码片段,如下所示。

```
1 start=System.currentTimeMillis();                                                       //获取起始时间
2 for(int i=0;i<10000;i++){                                                                //循环进行 10000 次检索
3     //每次执行检索前单独创建一个 PreparedStatement 语句
4     PreparedStatement psTemp=con.prepareStatement("select * from student where
stuNo=1005808");
5     ResultSet rs=psTemp.executeQuery();                                                  //执行检索
6     rs.close();                                                                           //关闭结果集
6     psTemp.close();                                                                      //关闭 PreparedStatement 语句
8 }
9 end=System.currentTimeMillis();                                                         //获取结束时间
```

这段代码的错误之处就在于每次循环中重复创建 PreparedStatement 对象,这么做根本就没有将预编译语句的优势发挥出来。PreparedStatement 效率高的代价是其在创建对象时付出较大的成本,但是对象创建后就可以通过传入不同的参数重复高效地使用。所以上述代码最耗时的地方就是第 3 行创建预编译语句的代码,而不是执行检索的代码。

四种使用 Statement 和 PreparedStatement 语句的方法耗时对比结果如图 10-15 所示,由图中可以看出错误使用 Statement 语句造成重复创建对象虽然增加了开销,但是由于 Statement 对象创建成本较低,所以两种使用 Statement 语句的差别并不是很大。

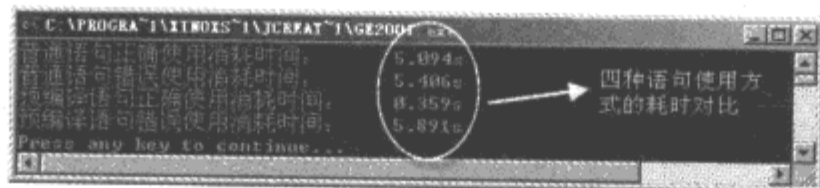



图 10-15 四种语句执行耗时对比

而错误使用 PreparedStatement 的后果将是执行效率的大幅降低,比其正确使用情况,速度相差了将近 20 倍,甚至比错误使用 Statement 语句的执行速度还要慢许多。


 **提示** 如图 10-15 所示的消耗时间根据不同的机器性能、不同的数据库（本例使用的是 JDBC-ODBC 桥连接的 Access）会有所偏差，但是正确使用 PreparedStatement 可以在一定程度上提高性能是基本不会变的。笔者的机器配置是目前主流的奔腾双核，相信读者朋友的机器性能应该会更好。

Statement 不是经过预编译的，所以一个 Statement 语句可以执行很多种完全不相关的数据库任务，如插入和查找。而 PreparedStatement 经过预编译处理效率有了很大的提高，但一个 PreparedStatement 只能执行一种类型的任务，例如接收学生证号查找学生信息、接收手机号查找计费信息等。

另外要特别注意的是，PreparedStatement 对象的创建成本高。因此如果不能正确地使用预编译语句可能会导致性能更差，还不如使用 Statement。

如何正确使用 PreparedStatement 呢？首先系统中需要存在这类需求：系统总是会较频率地反复执行同一类大业务量的数据处理任务，如输入学号返回考试成绩等。这种相似任务可以将其封装到一个 PreparedStatement 对象中，每次执行这个任务的时候，只需要传递相应的参数即可。

有没有多个相似的任务共享一个 PreparedStatement 对象，这是判断是否正确使用 PreparedStatement 的标准。虽然在本小节的举例中性能上的差异并不是很大，但是需要知道在对待服务系统尤其是大型服务系统时，对于性能再锱铢必较都不为过。

 **提示** 另外需要说明的是，PreparedStatement 需要数据库及 JDBC 驱动的支持，如果数据库或 JDBC 驱动不支持 PreparedStatement，则可能使用 PreparedStatement 后性能并不会有什么提升。例如，MySQL 目前在这方面的支持就不是很好。但是，使用 PreparedStatement 后就算性能得不到提升也会带来诸如避免 SQL 注入攻击之类的好处，有兴趣的读者可以研究研究。

### 10.3.3 管理数据库连接不知连接池

数据库连接相信每个开发人员都不陌生，也都操作过，但是数据库连接池估计就会有一些人不太熟悉了。如果一名 Java 开发人员尤其是 Java EE 开发人员不了解数据库连接池的话，那么或者其开发的项目都是小型应用，或者不对性能做较高要求。

“蔡佳娃，我说你看完没有啊，该去吃饭了。”

“等等，师兄，我很快就把你的‘stain’文件夹里的东西看完了。师兄，这里面写的‘数据库连接池’是干什么的啊？我怎么没听过啊？”

“看看，孤陋寡闻了吧，跟我当初一样，为防止这个成为你以后的‘污点’，由我来提前给你说个明白吧。”

首先介绍一下为什么要有数据库连接池。在进行数据库操作的时候，不管是对于访问请求者还是数据库服务器获取数据库连接的过程开销是非常大的。很多时候双方花费相当长的时间去创建连接，实际处理数据的时间却短得可怜。

一个数据库服务器运行起来，同一时刻所能提供的连接数也是非常有限的。如 Oracle 一般就是 50 左右，但是并发访问的数量却远不止这个数。如果请求者拿到数据库连接后，不慌不忙地进行操作，等到全部工作完成后再释放连接，这种情况下大量的请求者就会因为得不到连接而被堵



在外面，数据库服务器对于大量的连接请求也很难吃得消，如图 10-16 所示。

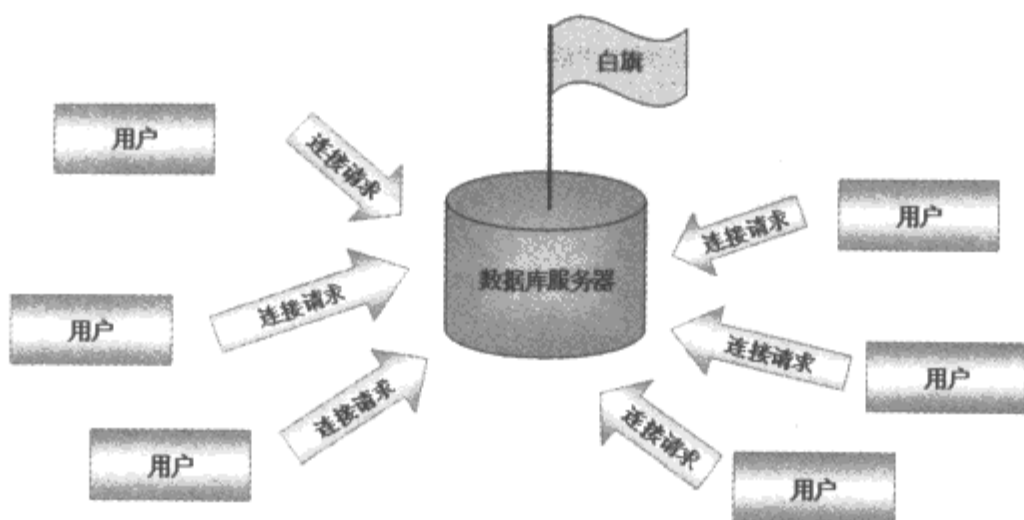


图 10-16 数据库服务器连接请求图

针对数据库服务器不堪重负的情况，就产生了“随开随关”的处理方式。“随开随关”的原则就是“最晚开，最早关，需要时再开”。这样尽可能地缩短用户与数据库服务器的连接时间，使其能够高效率地处理其他访问者的请求。

这种做法的确是解放了数据库服务器，但同时也将数据库访问者带进了长时间等待的黑暗之中。因为对于每一次的数据库连接，都要创建一个 Connection 对象，其创建成本很高，时间相对漫长，一次 Connection 对象的创建，大约可以执行 8 到 10 次甚至更多的数据库检索。

这时如果充分发扬“随开随关”的精神，那么用户将会感觉非常慢。因为大量时间都花费在创建 Connection 对象上，即使数据库服务器有连接可用，也会因为创建 Connection 对象而使得自己步履蹒跚。同时 Web 服务器也会因为频繁地生成数据库连接对象而使得并发性能下降，如图 10-17 所示。

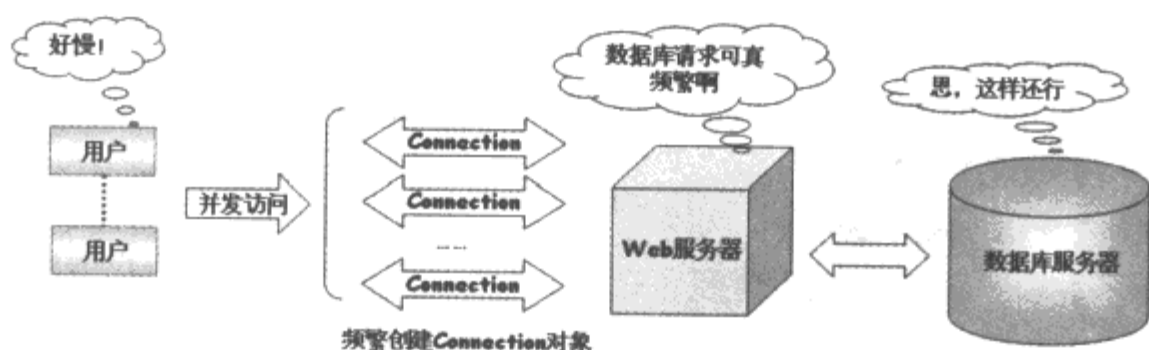


图 10-17 “随开随关”造成客户端性能下降

这时就陷入了两难的境地，一方面如果让用户慢条斯理地使用数据库连接则数据库服务器会瘫痪掉，而严格限制用户使其“随开随关”又会让用户慢得无法忍受并加大 Web 服务器开销，这时数据库连接池就提供了一个比较完美的解决方案。

数据库连接池的原理是这样的：为服务器（如 Tomcat）配置一个连接池，配置文件中指定所要连接的数据库和连接个数，这样服务器启动的时候，首先会连接目标数据库创建指定个数的活动连接，并将其放到连接池中。

这时候数据库访问者遵循的就是“最晚借，最早还”的原则了，如果 Web 或其他应用中需要连接数据库，则从连接池中“借”一个过来，通过这个连接进行相关操作，当用完之后再将其“还”回去。这样不同访问者共享了有限的连接，免去了数据库连接的漫长创建过程，使得性能大大提

高，如图 10-18 所示。

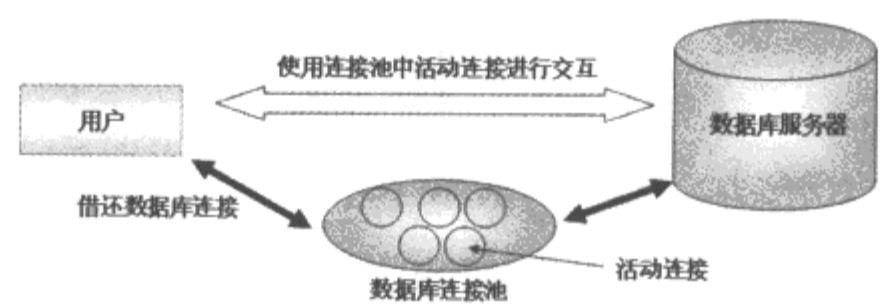


图 10-18 数据库连接池原理图

使用数据库连接池对于服务器的好处是最大的，下面来对有无数据库连接池两种情况的服务器性能做一个对比。假设某个数据库服务器同一时刻只提供 3 个连接，在某个时刻有 5 个数据库访问请求到达，如果不采用数据库连接池，则服务器处理完这 5 个访问所需要的时间如图 10-19 所示。

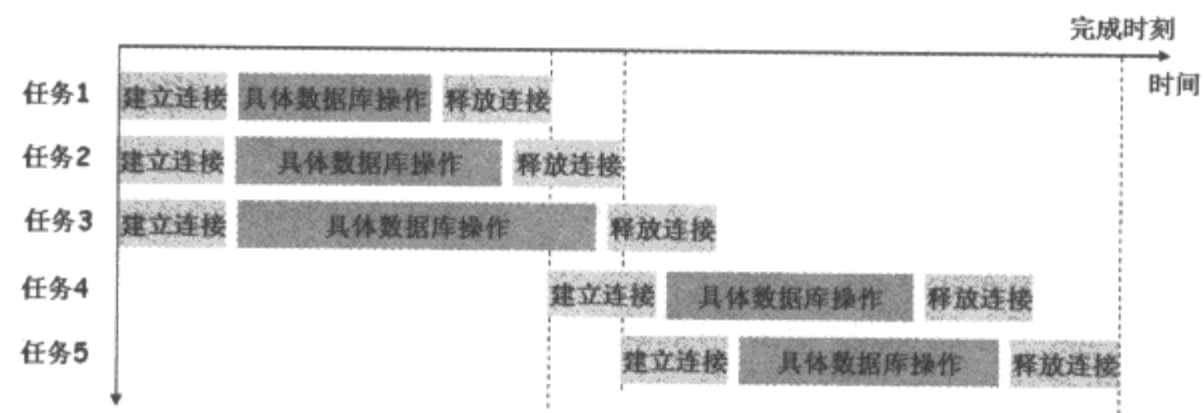


图 10-19 不使用连接池管理数据库连接

从图 10-19 中可以看出，任务 4 和任务 5 必须等到任务 1 和任务 2 处理完成释放连接后才可以建立连接进行自己的操作。服务器处理完毕的时刻是任务 5 完成的时刻，若采用数据库连接池处理连接，则服务器需要处理的时间如图 10-20 所示。

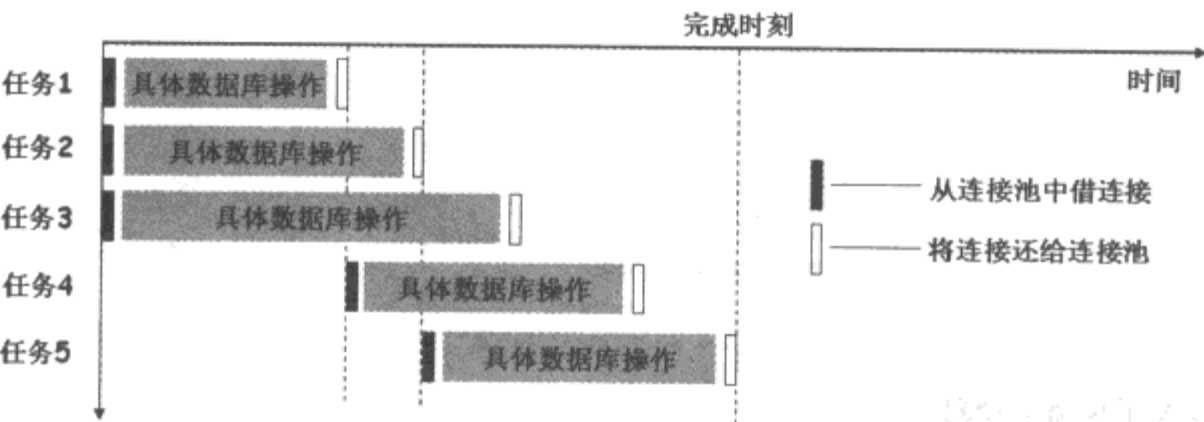


图 10-20 使用连接池管理数据库连接

虽然具体的数据库操作时间没有改变，但是从连接池获取连接而不是创建连接使得所有任务的总处理时间大大减少。如果一台服务器（如 Tomcat）上只运行了这一种服务程序（如 Web 应用），那么这个程序的性能将会得到很大的提升。

但一般情况下一台服务器上都会有很多程序同时运行，这时采用数据库连接池不仅可以使自己的程序执行效率更快，而且与不采用连接池的做法相比，节省出了很多的系统资源（如图 10-21 所示），使得服务器可以更有效地去运行其他应用程序。

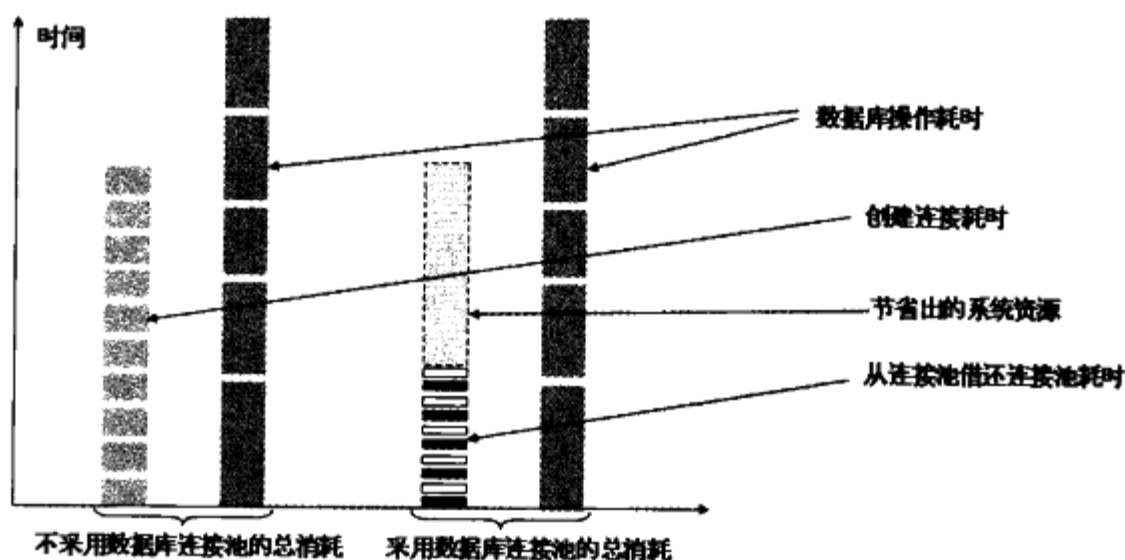


图 10-21 有无数据库连接池消耗总资源对比

数据库连接池的原理虽然复杂，但使用起来却很简单。为了加深理解，本书以在 Tomcat 中配置一个 MySQL 数据库连接池为例，详细步骤介绍如下。

① 在 Tomcat 安装目录下 conf 目录中找到 server.xml，在最后的“</Host>”标记之前添加如下代码：

```

1  <Context path="/TC_DSTest" docBase="C:/webapps/TC_DSTest"
2      debug="5" reloadable="true" crossContext="true" workDir="">
3      <Resource name="jdbc/TC_DSTest"
4          auth="Container"
5          type="javax.sql.DataSource"
6          maxActive="100" maxIdle="30" maxWait="10000" <!-- 活动连接数等 -->
7          username="" password=""
8          driverClassName="org.gjt.mm.mysql.Driver" <!-- 驱动类 -->
9          url="jdbc:mysql://localhost/test"/> <!-- 数据库连接 URL -->
10 </Context>

```

上述代码为 MySQL 中的 test 数据库配置了一个连接池，其中连接池内最大连接数为 100，理想连接数为 30，最大等待时间为 10 秒（10000 毫秒）。

② 在本应用的 WEB-INF 目录下 web.xml 文件中添加如下配置。

```

1  <resource-ref>
2      <description>DB Connection</description> <!-- 资源描述名称 -->
3      <res-ref-name>jdbc/TC_DSTest</res-ref-name> <!-- 资源 JNDI 名称 -->
4      <res-type>javax.sql.DataSource</res-type> <!-- 资源类型 -->
5      <res-auth>Container</res-auth>
6  </resource-ref>

```

③ 在 JSP、Servlet 或 JavaBean 中用如下 Java 代码获取数据库连接。

```

1  Context initial = new InitialContext(); //创建初始上下文
2  DataSource ds = //通过 JNDI 获取数据源
3      (DataSource)initial.lookup("java:comp/env/jdbc/TC_DSTest");
4  Connection con=ds.getConnection(); //从连接池中借一个连接

```

若要切换数据库，则只需要改动第一步中的配置文件即可，这也增强了系统的可维护性，使用数据库连接池有如下几个方面的好处。

- 提升性能

这是最明显的优点，使用数据库连接池后，用户进行数据库操作时只需要向连接池借来连接引用即可快速进行操作。这样使得有限的资源被最大化地利用，将数据库服务器和终端用户的性能需求协调得恰到好处。

其实，在电信业中也是采取这种类似“借还”的原则。比如一个小区有 500 家住户，每户都有固话接入，但是因为同一时刻并不是每个人都在打电话，真正交换机的外线出口只有 10 到 20 条，这样既满足了用户的需要，也将电信的负载降到了最低。

- 安全性更好

在没有连接池的情况下，开发人员需要打开数据库连接进行操作时，必须要被告知连接字符串、数据库用户名和密码等相关信息，这样机密的信息被掌握在多数人手中，当然不利于系统安全。

同时这种管理方式也容易受到外部攻击，攻击者很容易从 Web 层获取数据库层的核心管理信息，一旦攻击者获得了数据库管理权限，那么整个系统就在攻击者面前一览无遗了。

而如果使用了数据库连接池，开发人员只需要被告知连接池的联系方式即可。至于连接池中的数据库连接是如何创建的，开发人员没有必要知道，只负责好借和还就行了。这样将数据库连接的详细信息对外界透明化，使得数据库更加安全。

- 更容易管理

如果采用“最晚开，最早关”的原则去管理数据库连接，性能的问题不谈，一个后果就是将请求数据库连接的代码写得到处都是。假设某一天需要从原来的 Access 数据库转为 MySQL 或 Oracle 数据库，那么原来打开连接的地方，都要将连接字符串等进行修改，其工程的烦琐程度可想而知。

而采用了数据库连接池技术，只需要在配置文件中修改相应数据库驱动和连接字符串，重启一下服务器，就完成了数据库切换的工作，几乎都没有什么工作量可言。最重要的是这种修改不需要涉及源代码，测试起来就容易多了。

**提示**

“池化”是一种软件开发中非常重要的思想，不仅可以用在数据库连接的管理上，在很多地方也都有应用。例如在线程管理中为避免线程资源的浪费和提高效率，也可以采用线程池来进行线程的管理，在 Java SE 6 中还专门提供了这方面的 API。

## 10.4 忽视内存管理

了解 Java 的开发人员都应该知道，Java 相比于 C++ 的一个优势就是采用了垃圾收集器的方式自动管理内存，这样在一定程度上减轻了开发人员的负担。

“师兄啊，我越来越发现用 Java 开发的好处了。尤其是垃圾收集器，多么像一个勤劳忠实的仆人啊，我根本不用关心内存问题，它会自动帮我打扫卫生。”

“打住，这话可不对啊。垃圾收集器绝对不是仆人，应该算是助手。Java 的内存管理需要开发人员的配合，绝对不是那么轻松实现的。”

“不是吧？还需要我们插手吗？”

打个比方来说，Java 的内存管理只提供相当于物业公司的服务，并不是保姆，垃圾管理器是不会擅自闯入家中将垃圾倒掉的。开发人员需要做的就是标识什么是垃圾，垃圾收集器才会将其回收。否则就算户主的东西堆满楼道，物业公司也只是敢怒不敢扔。

### 10.4.1 对象的 3 种引用

关于 Java 内存管理机制以及垃圾收集器的工作原理很多书上都有介绍，如《Java SE 6.0 编程指南》等，有兴趣的读者可以详细阅读。本节来介绍一些平时不为大家注意的内存管理技术，首先就是对象的 3 种特殊引用。

正常情况下声明一个引用并将其指向一个对象，等到对象没有存在价值的时候将引用拿掉，这样的话垃圾收集器就会在某个时刻将该对象的内存回收。但是这种简单地将对象分为“有用”和“没用”的情况并不适合于所有的实际场合，因为开发过程中有很多介于“有用”和“没用”之间的“灰色地带”。

这种情况下采用传统的对象引用方式就不能完全满足要求了，这时就应该采用弱引用、软引用或幻影引用了。

#### 1. 弱引用

弱引用是这样一种引用：当被弱引用指向的对象除该弱引用之外不再有其他引用指向这个对象时，这个对象就是“弱可达”的。这时的对象仍然可以通过弱引用访问，但其名义上已经是垃圾了，只要垃圾收集器检查到这个弱引用指向的对象，就会将其收回，如图 10-22 所示。



图 10-22 弱引用指向的对象被垃圾收集器回收

Java 中提供了一个 `WeakReference` 类帮助实现弱引用，它位于 `java.lang.ref` 包下。`WeakReference` 类中有一个方法：`public Object get()`，可以通过该方法获得弱引用指向的对象。下面给出一个使用弱引用 `WeakReference` 的例子，代码如下所示。

```
1 package wyf;
2 import java.lang.ref.*;           //引入相关包
3 //定义被弱引用指向的类
4 public class WeakRef_Sample{
5     public void shout(){           //自定义方法，表明该对象仍然存在
6         System.out.println("哈哈，我还活着!!");
7     }
8     public void finalize(){        //重写 finalize 方法，该方法在对象被回收时自动调用
9         System.out.println("啊哦，我被垃圾收集器回收掉了!");
10    }
```



```

10     }
11     public static void main(String args[]){
12         WeakRef_Sample sample = new WeakRef_Sample();//创建将要弱应用指向的对象
13         WeakReference wr = new WeakReference(sample);//定义弱引用并指向创建的对象
14         sample = null;                                //使对象变成垃圾
15         ((WeakRef_Sample)wr.get()).shout();           //通过弱引用访问垃圾对象
16         System.out.println("系统开始执行垃圾收集!");
17         System.gc();                                  //执行垃圾收集
18         try{                                          //让主线程休眠,是垃圾收集器运行
19             Thread.sleep(80);
20         }
21         catch(Exception e){
22             e.printStackTrace();
23         }
24         if(wr.get() != null){                        //验证是否还能访问弱引用指向的对象
25             ((WeakRef_Sample)wr.get()).shout();
26         }
27     }
28 }

```

**提示** 由于在使用弱引用访问垃圾对象的时候不保证其在内存中的存在,所以需要如上述代码的 24 行那样进行判断后再使用。

上述代码运行后的结果如图 10-23 所示。

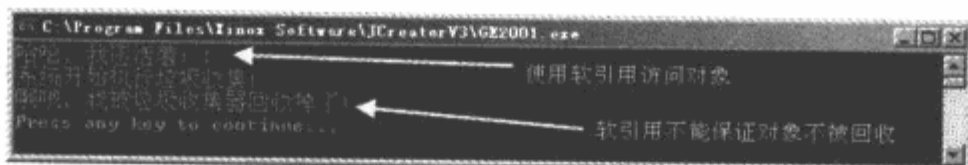


图 10-23 弱引用示例运行结果

弱引用可以使得被判定为垃圾的对象仍然可以被访问到,这样可以满足某些情况下希望无用对象继续保留以备后用的需要。但是弱引用并不提供完全的保证,一旦垃圾收集器运行并检测到该弱引用指向的对象,会毫不留情地将其回收。

弱引用最大的用途是用来防止内存泄露,举一个例子来说,开发一个房间类的局域网游戏,需要将所有创建的房间对象进行统一管理,一般的解决方案是在房间创建后将其保存到一个 HashMap 中。

但是问题也随之而来,当一个房间的游戏结束后,自然会将房间对象设置为垃圾。但是如果忘记从管理房间的 HashMap 中删除该房间,导致仍有引用指向该房间对象。于是在整个游戏服务器运行时间内,这个废弃的房间对象将会一直保留在内存中,无法被回收,如图 10-24 所示。

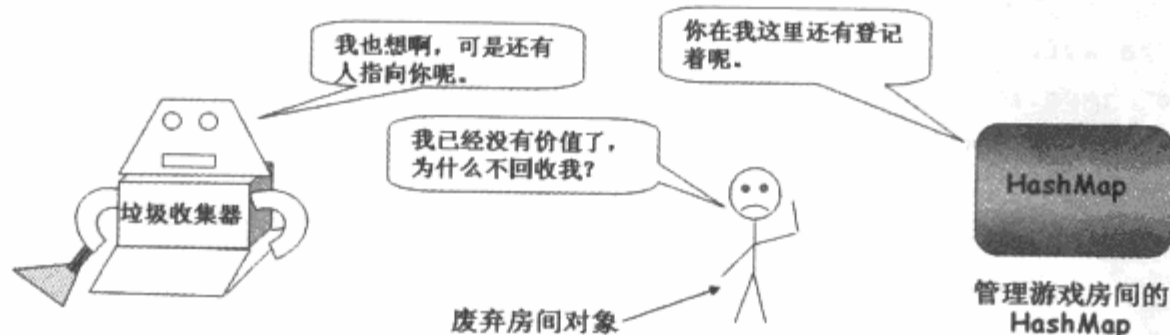


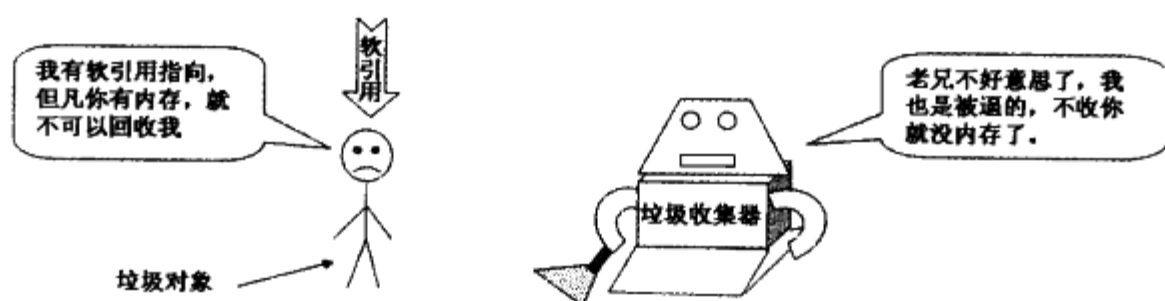
图 10-24 内存泄露原理

针对这种隐患，使用弱引用就可以解决这个问题，这里需要提到一个管理弱引用的集合：`WeakHashMap` 类，它与 `HashMap` 类不同的地方在于 `WeakHashMap` 中键值对象的引用一律为弱引用。其他的用法和 `HashMap` 类基本相同。

针对前面提到的房间类游戏的问题，如果管理游戏房间的集合使用 `WeakHashMap` 的话，在房间游戏进行时，垃圾收集器不会将其定为垃圾。当房间游戏结束后，由于除 `WeakHashMap` 中的弱引用之外，该房间对象已经不存在任何引用，将会被垃圾收集器收回，避免了内存泄漏。

## 2. 软引用

软引用是这样一种引用：当被软引用指向的对象除该软引用之外不存在其他的引用指向这个对象时，这个对象就是“软可达”的。“软可达”的对象将会一直保留在内存中，除非内存被耗尽，否则垃圾收集器都不会将其回收。在内存资源接近崩溃的时候，垃圾收集器才会将软引用指向的对象回收，如图 10-25 所示。



10-25 软引用指向的对象被垃圾收集器回收

Java 中软引用的实现需要 `java.lang.ref` 包下的 `SoftReference` 类支持，下面给出一个使用软引用保留垃圾对象的例子。

```

1  package wyf;
2  import java.lang.ref.*;           //引入相关包
3  //创建软引用指向的对象
4  public class SoftRef_Sample{
5      public void shout(){           //自定义方法，表明该对象仍然存在
6          System.out.println("哈哈，我还活着!!");
7      }
8      public void finalize(){        //重写 finalize 方法，该方法在对象被回收时自动调用
9          System.out.println("啊哦，我被垃圾收集器回收掉了!\n");
10     }
11     public static void main(String args[]){
12         SoftRef_Sample sample = new SoftRef_Sample();//创建将要被弱引用指向的对象
13         SoftReference sr = new SoftReference(sample);//创建软引用对象
14         sample = null;               //将对象置为垃圾
15         ((SoftRef_Sample)sr.get()).shout();           //验证对象是否还存在
16         System.out.println("系统开始执行垃圾收集!");
17         System.gc();
18         try{
19             Thread.sleep(80);        //让主线程休眠，使垃圾收集器运行
20         }
21         catch(Exception e){
22             e.printStackTrace();

```

```

23         }
24         if(sr.get() != null){                                //验证对象是否还存在
25             ((SoftRef_Sample)sr.get()).shout();
26         }
27         String [] buf = new String[10000000];                //定义一个大数组，使其耗尽内存
28         for(String s:buf){
29             s = new String("我就是来让内存崩溃的。");
30         }
31     }
32 }

```

上述代码运行的结果如图 10-26 所示。系统在内存超出后抛出 `OutOfMemoryError` 异常，在这之前会将所有软引用指向的对象通过垃圾收集器回收。运行命令中 “-Xmx20m” 参数的意思是告诉虚拟机此次运行分配给程序的最大内存为 20MB。

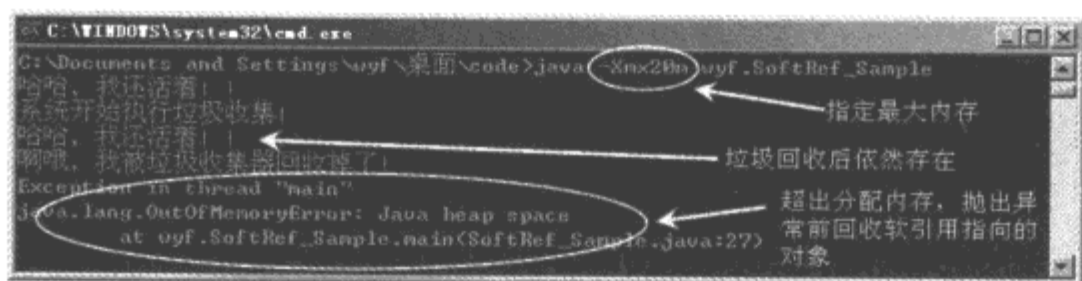


图 10-26 软引用示例的运行结果

相对于弱引用来说，软引用在保留不用的对象方面显得更为有效，只要内存还有富余，垃圾收集器就不能将废弃对象回收。只有在内存渐渐吃紧将要耗尽的时候，垃圾收集器才会在抛出异常之前尝试回收这些软引用指向对象的内存。

**提示** 软引用指向的对象被回收并不是总伴随着内存耗尽异常的抛出，如果释放部分软引用指向的对象之后系统内存可以正常使用，则不会抛出 `OutOfMemoryError` 异常。

软引用最大的用途就是用来做缓冲区，在传统需要缓冲区（一般是 `byte` 数组）的程序开发中，一般有两种方案。第一种是需要的时候再申请内存，第二种是在使用之前如创建对象时就申请好，以后的程序都使用这个缓冲区。

第一种方案在缓冲区的访问频度很高时会对系统性能有很大的影响，而第二种方案则会让缓冲区在内存中的存活时间延长，由于存在有效引用指向，垃圾收集器宁可抛出内存耗尽异常都不会去尝试回收这些缓冲区中的对象。

而此时使用软引用就非常实惠了，在对象创建时就将缓冲区初始化，并用软引用指向这个缓冲区对象，这样一来保证了缓冲区可以重用以提高系统性能。同时如果系统内存吃紧则可以让垃圾收集器将其所占用的内存收回，这样既避免了内存资源的浪费，也在一定程度上使系统执行效率更快。

### 3. 幻影引用

幻影引用又称为虚引用，幻影引用不同于软引用，它所指向的对象和一般垃圾对象一样会被回收，并不能延长存活时间。

幻影引用也不同于弱引用，幻影引用的一个特点就是其必须和一个 `ReferenceQueue` 类一起使

用。当一个被幻影引用指向的对象被垃圾收集器检测到需要回收时，垃圾收集器会将其挂到 ReferenceQueue 队列中，这样可以起到类似消息传递的作用，使对象内存在被回收前执行指定的 pre-mortem 操作（pre-mortem 可以理解为死前）。

幻影引用所涉及的知识超出了本书的讨论范围，本节主要侧重于前两种对象引用的特性和使用，对于幻影引用有兴趣的读者可以自行参阅其他资料进行研究。

#### 10.4.2 “小肥猪”问题

“小肥猪”问题泛指内存管理中一个非常严重而又极易出现的问题，这类问题多发生在企业级服务器程序中。这里的“小肥猪”是开发人员由于疏忽产生的有害代码，“小肥猪”专以内存资源为食，如果不小心将其放到程序中，那么“小肥猪”就会永远不知饱地将内存吞噬殆尽。

“蔡佳娃，刚才我们谈的是基于 Java 自身的内存管理技术如垃圾收集器来进行内存管理，这些知识需要引起每个开发人员的重视哦。”

“嗯，师兄，我会记住的。”

“好，我们下面来谈谈软件开发时经常会产生的一种内存问题。很多时候，这个问题的产生都是由于开发人员对内存管理的忽视造成的，这个问题就是‘小肥猪’问题。”

“‘小肥猪’问题？那是什么问题啊？”

“‘小肥猪’一般出现在企业级服务器程序中，这些服务程序在启动的时候会创建并加载一些对象，而这些对象在服务器运行中会产生新的对象。这些新产生的子对象如果没有及时在用完之后释放掉，就会依附在创建它们的对象上，这个被依附的对象就成了‘小肥猪’。”

“师兄，你讲得好抽象啊！我不太明白。”

“呵呵，好吧，我们来举个例子你就明白了。”

假设 Web 服务器启动后，会创建一个 PreparedStatement 对象，这个对象负责处理这样一种数据请求：用户输入手机号码，服务器则返回话费清单。根据本章前面介绍过的知识，这种情况下使用 PreparedStatement 会提高系统效率。

但是在这种情况下 PreparedStatement 对象是被多个客户共享的，它伴随着服务程序的运行一直存在，多个客户端共享同一个 PreparedStatement 对象进行数据库的访问如图 10-27 所示。

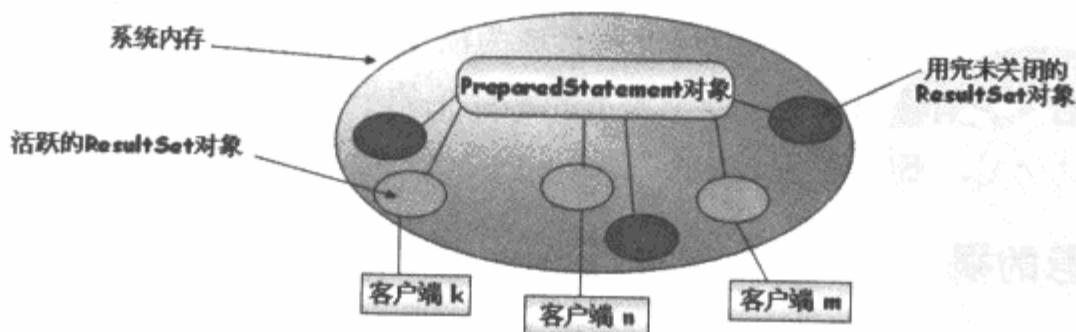


图 10-27 PreparedStatement 被共享图

图 10-27 中的 ResultSet 对象是在本例中数据库进行手机号码查询后产生的包含话费清单的结果集，虽然多个用户共享 PreparedStatement 对象，但是每个用户都会分配一个唯一的 ResultSet 对象。

当处理完用户请求之后，这些 `ResultSet` 对象应该被关闭，如果因为疏忽没有将其关闭，那么就会产生“小肥猪”问题了。由于 `PreparedStatement` 会在服务器运行期间一直存在，其产生的那些未关闭的结果集也就会长期伴随存在，并且越积越多，“小肥猪”也越来越肥，如图 10-28 所示。

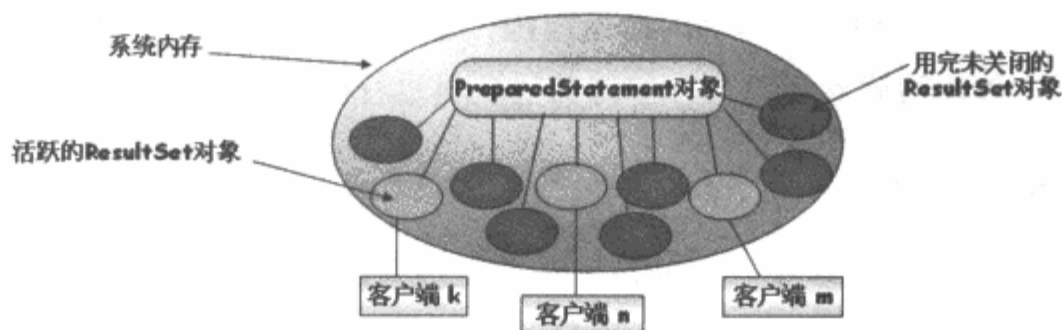


图 10-28 “小肥猪”问题

于是系统就会出现这种状况，一开始运行的时候执行速度很快。但是随着时间的推移，系统开始变慢，几个礼拜或一个月之后，系统内存可能就会崩溃。这个时候如果重启服务器，速度慢的症状就会消失，因为已经进入了下一轮“小肥猪”喂养的循环。

这里的“小肥猪”很像古代的神兽貔貅，貔貅只有嘴无肛门，能吞万物而从不泻，可招财聚宝。正所谓“家有貔貅，万事无忧”，现在很多中国人配带貔貅的玉制品正因如此。“小肥猪”在“只进不出”这个特性上颇似貔貅，但是对于 Java 开发界，这可绝对不是什么吉祥物，一定要坚定不移地杜绝“小肥猪”现象的发生。

除了本例中不当使用 `PreparedStatement` 会造成“小肥猪”问题外，还有许多可能产生“小肥猪”的隐患，比如服务器基于 `Socket` 和客户端进行连接时，如果不注意关闭输入输出流的话，也可能使得无用对象慢慢蚕食系统内存，造成资源的严重浪费。

“小肥猪”问题是比较大型的 Java 应用开发时容易犯的错误，但是如果考虑周到，小心谨慎一些，这种现象还是很容易避免的。

## 10.5 看了就不要再犯的错误

要想成就一些大型应用软件，必然依靠的是复杂的设计和高深的思想，但是“九层之台，起于累土”，不管多么巨大的问题，最终都是随着一个一个简单问题的解决而解决的。如果对这些简单问题一知半解，思考不够彻底，很有可能就会成为溃千里之堤的“蚁穴”。

在本章的最后，我们将列举一些开发人员经常会犯的小错误，虽是小错误，但小错误有大危害，希望读者引以为戒，不要再犯此类的错误。

### 10.5.1 “+”惹的祸

这里的“+”号特指 `String` 对象在进行字符串连接的时候进行的“+”操作，`String` 对象的这种操作使用起来很简单，但是性能上却有着很大的劣势。很多人习惯于用加号来进行字符串的连接，以至于在不知不觉中给系统的效率埋下了巨大的隐患。

“师兄，我最近翻书居然翻出自己当年考 SCJP 证时的复习资料，重新看一遍，发现仍然受益匪浅啊，很多当时记住的东西时间久了就全忘了。”



“是啊，练戏曲的不是有口诀嘛：一天不练手脚慢，两天不练丢一半，三天不练门外汉，四天不练瞪眼看。所以开发人员手不能懒，要温故而知新。”

“是啊，这次重新翻阅当年的资料，印象最深的是有关字符串对象的问题。如果当年对‘字符串是常量’这个概念有更深入的理解，现在对字符串进行操作肯定更加自如啊。”

字符串在内存中的存储机制很特殊，并且一旦诞生内容就永不改变。当对 String 对象进行连接时，其实是在创建很多无用的中间对象，这样大大增加了系统的开销，并导致性能的严重下降。下面给出一个分别使用 String 和 StringBuffer 对字符串进行连接的性能对比的例子，代码如下所示。

```

1  package wyf;
2  public class Cmp{
3      public static void main(String args[]){
4          String s = new String();           //定义 String 对象
5          long start = System.currentTimeMillis(); //获取开始时间
6          for(int i=0;i<10000;i++){          //对字符串进行 10000 次连接
7              s = s+i;
8          }
9          long end = System.currentTimeMillis(); //获取结束时间
10         System.out.println("使用 String 对字符串进行连接耗时: \t" + (end-start) + "ms");
11                                         //打印输出
12         StringBuffer sb = new StringBuffer(); //定义 StringBuffer 对象
13         start = System.currentTimeMillis(); //获取开始时间
14         for(int i=0;i<10000;i++){          //对字符串进行 10000 次连接
15             sb.append(i);
16         }
17         end = System.currentTimeMillis(); //获取结束时间
18         System.out.println("使用 StringBuffer 对字符串进行连接耗时: \t" + (end -
19 start) + "ms");
20     }
21 }

```

上述程序的运行结果如图 10-29 所示，String 和 StringBuffer 执行相同的工作，效率相差超过 100 倍。所以在进行大量字符串连接的时候，一定不要用 String 对象，而应使用 StringBuffer 对象，二者操作差不太多，但是性能却差之千里。

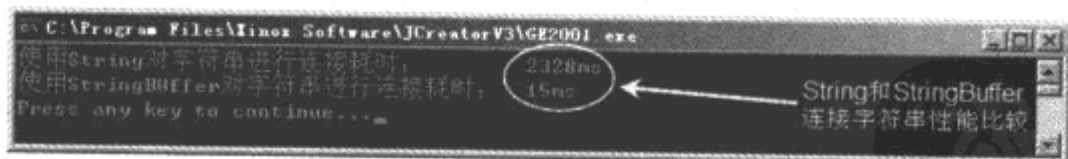


图 10-29 String 和 StringBuffer 连接字符串性能比较

**提示** 从 Java SE 5 开始，类库中针对这个问题又提供了一个 StringBuilder 类，在不考虑多线程并发的情况下，StringBuilder 会比 StringBuffer 有更好的性能。

值得注意的是，有些情况下对 String 对象进行的“+”操作并不会降低系统性能，请观察以下代码。

```

1  String s = "Hello " + "World!" + "Welcome " +
2      "to " + "the " + "Java " + "World!";

```

上述代码虽然也大量使用了加号，但是在运行的时候，内存中只有一个字符串对象，即“Hello World! Welcome to the Java World!”。这是由于 Java 的编译器很聪明，编译器扫描到赋值表达式右边全是字符串常量后，会将这些常量变成一个整字符串，这样编译运行的时候是以一个字符串的身份被访问的。所以说这样分开写是没有错的，还可以在一定程度上增加代码的可读性。

但需要注意的是要想让编译器“Smart”起来，必须保证赋值表达式右边都为字符串常量，以下代码就说明了这个问题。

```
1  String s1 = "Java ";           //为字符串赋初值
2  s1= s1+"Hello ";              /*
3  s1= s1+"World!"               * 不断进行字符串的
4  s1= s1+ "Welcome "           * 连+操作，编译器想
5  .....                        * smart 都难
3  s1=s1+ "World!";             */
```

这种情况下由于在等号右边出现了 String 类对象的引用 s1，则在编译的时候编译器不会将其在运行前整合为一个字符串。这些字符串变量将以对象的形式单独存放在内存中，因此虽然最终效果相同，但第二段代码的效率要比第一段低很多。

## 10.5.2 魔法数字

魔法数字最直白的解释就是一个确切实在的普通字面常量值，当这个字面常量值在程序中的很多地方都被用到时，它就成为了魔法数字。魔法数字的魔法效果发生在这个数字因为实际需要或其他原因需要对值进行改变的时候。

当需要对魔法数字进行修改时，繁重的工作不说，往往会因为修改不完全造成程序出现意想不到的魔法效果。不过这些魔法效果一般不会是哈利波特经常用的那种好看的魔法，而是如伏地魔那般邪恶的黑魔法，很容易让人头疼抓狂。

“蔡佳娃，你以后写手机上的小游戏，不要再拿给我看了，每次都是小毛病，太影响你在我心目中的美好印象了！”

“啊，师兄，不要这样啊，你对我应该‘不抛弃，不放弃’才行啊！”

“就拿这次来说吧，你看看你的代码里面，通篇都是眼花缭乱的数字。什么 20、25 之类的，又代表图片大小，又代表移动步长的，我花了足足半个小时，才整理出一张你的魔法数字和真实含义之间的映射表。”

“师兄你教训得很对啊，我当时急着开发，所以注释全没写，那些尺寸之类的数字也是随用随写，代码的确读起来很难理解啊。”

实际开发中使用魔法数字不仅会影响到其他人阅读自己的代码，而且随着时间的推移自己也会将其淡忘。那样的话这段代码除了作为千古之谜，已经没有任何价值了。

这只是魔法数字带来的一种轻量级的危害，如果所开发的项目中有大量的魔法数字，当需要对其进行改动时，则需要将每个出现魔法数字的地方进行修改，这种重复且无价值的工作很多时候都让人无法忍受。万一在大规模改动过程中出现了细微的错误，那么查找错的过程就是魔法数字对开发人员施放的最大梦魇。

所以在开发中要尽量避免使用魔法数字，一旦发现某个字面常量在程序中多次出现，应该将其定义为一个变量。如一个计费软件的费率因子、游戏中屏幕的尺寸等。将魔法数字变量化可以使得程序在需要改动时做到“牵一发而动全身”。

不仅如此，字面常量变量化在调试程序的时候也非常有用，如将循环次数、线程休眠时间等跟调试有关的信息定义为变量，可以更方便地进行更改、调试。


“哦，师兄我明白了，我保证下次再给你看的程序中不出现魔法数字！”

“其实呢，不仅常量类的数值需要声明，很多时候魔法数字还包括数值以外的类型，比如字符串。”

“字符串？字符串能有什么魔法啊？”

“就比方说你这次给我的手机游戏，游戏中很多地方需要连接 Socket 服务器，我就发现你的连接字符串总是随用随写，这么一来万一你的 Socket 要有如 IP 地址之类的变动，那么你就要翻遍代码去修改那些连接字符串了。这不跟魔法数字一样嘛。”

“哦，我明白了，看来不仅有魔法数字，还有魔法字符串，我以后一定注意这方面的问题！”

 **提示** 上述字面常量变量化并不是指将字面常量定义为普通的变量，那样容易带来问题，而是指应该将字面常量定义为程序常量，如 C 语言中的“const”。Java 中虽然没有提供声明程序常量的方法，但可以通过将变量定义为 final 来实现程序常量，解决魔法数字的问题。

### 10.5.3 代码复制师的渺茫前途

本书的第 5 章在讲述菜鸟如何锐意进取的时候，提到了在看书的时候不要做代码观察师而是勇于实现书中的代码这个问题。本小节将介绍另一种前途渺茫的职业——代码复制师。

“蔡佳娃，我来突击检查一下，你平时工作中键盘上用得最多的键是什么啊？是不是 Ctrl+C 和 Ctrl+V 啊？”

“嘿嘿，是没少用，复制粘贴比较省力嘛。”

“我不是反对你用复制粘贴，只不过身为开发人员有些时候复制粘贴并不是解决问题的最好办法，下面我们就分情况讨论迷信复制粘贴的危害。”

平时的开发中应该并没有很多机会去直接复制，如果发现自己经常游离于 Ctrl+C 和 Ctrl+V 这两组快捷键之间，那么就应该好好审视自己是哪里的问题了。一般情况下，开发过程中出现的复制粘贴分为两种情况，下面将一一说明。

第一种情况是将代码从一个地方复制到另一个地方后不改一字，直接就能用。这种复制比较常见，相信不少人都这么干过，这种貌似代码重用的复制其初衷大概是为了省时省力。不过从长远来看，这根本算不上代码重用，而且当被大量复制的代码需要作修改时，就再也不会感觉到省时省力了。具体情况如图 10-30 所示。

所以在开发中如果发现某一段代码具备了“放之四海而皆准”的特性，不应该去复制粘贴，应该将这段代码封装起来作为一个方法。然后在需要这段代码的地方调用这个方法即可，这样才是真正的代码重用，而且需要修改时也会非常方便。

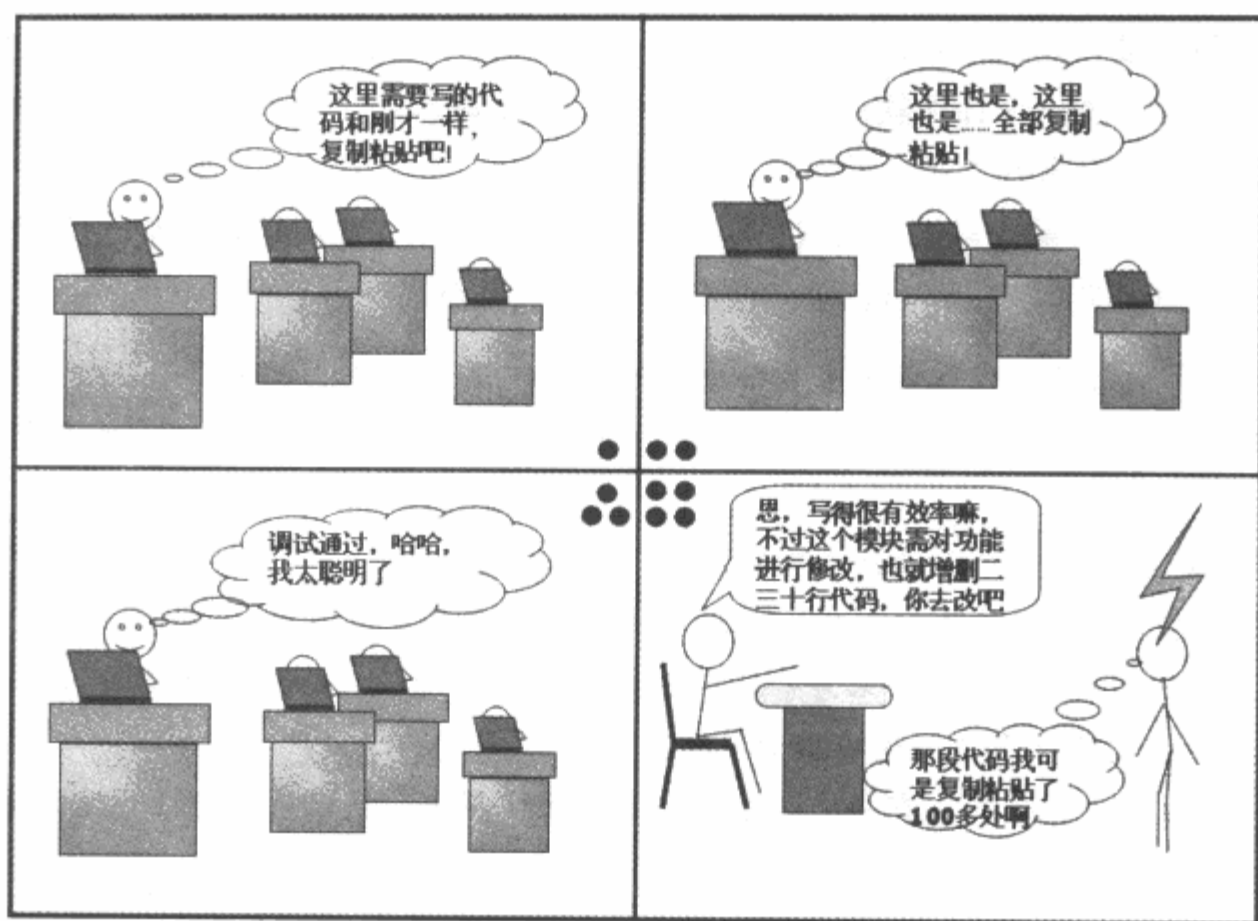


图 10-30 直接复制粘贴的弊端

魔法数字也属于一种代码复制，如果字面常量也可以称之为代码的话。不过魔法数字产生的危害仅仅是由于数字值的变化，而如果需要对被复制的大段代码做改动，绝对不是仅仅修改某个数值这么简单的，很可能需要对其进行大的手术，甚至颠覆原先的业务逻辑。

“蔡佳娃，看到第一种复制粘贴的厉害之处了吧！”

“是啊，师兄。那还有一种是什么情况呢？”

“不要着急，接着往下看，第二种不但有潜在问题，而且当时就可能遭报应哦。”

以上是一种复制粘贴的情况，还有一种复制粘贴的情况是需要编写的代码和某段已经存在的代码比较类似。这时候复制过来再进行修改应该会更快速一些，会提高开发效率。但是往往和前一种情况一样，这样做会收到相反的效果。

由于是针对复制来的代码作修改而不是从头写，在改的过程中可能就会遗漏本该修改的地方，这些遗忘点有时会在编译时期检查出来，但更多会逃过编译器而造成运行时期的 bug，这时就需要进行漫长烦琐的调试排错了。尤其是代码比较长时，其效果远不如重新开发，如图 10-31 所示。

权衡利弊，在遇到这种情况时最好还是不要复制，根据笔者多年的开发经验，在复制而来的代码基础上进行修改基本上都会出现遗漏的地方。而重新开发，虽然速度或许会慢一些，但是保证不会出现调试时的“无底洞”。

#### 10.5.4 老寿星变量

老寿星变量是指那些生存期大大超过其使用期的变量，这些长寿变量多是由于定义不恰当造成的。没有用武之地的老寿星变量只会占用系统资源，所以在开发中要注意杜绝。

“蔡佳娃，提到了魔法数字，我们再来谈谈另一个和变量有关的问题，那就是老寿星变量。”

“老寿星变量？”

“比如说一个变量定义为局部的就够用了，如果定义为类的成员变量或是静态变量，那么这个变量就会在使用完成后一直存在，直到对象被释放或程序运行结束。你给我的程序中就有一些变量属于这种老寿星变量。”

“哦，这个啊，不过区区一个变量应该不会对系统有什么影响吧？”

“怎么可以这么说呢？一个类中，除了变量就是方法，怎么能忽视呢？而且这里说的变量也不是简单的基本类型的变量，一个庞大的类也可以作为变量，如果它成了老寿星，那你说对系统资源的浪费是不是很严重呢？”

“对对，我忽略了 Java 中类的对象也可以做变量，这样一来老寿星的问题的确很严重啊。”

一个变量，应该根据需要对其生命周期进行量体裁衣，如果一个变量只在方法内部有效，就将其定义为局部变量，如果对整个对象有效，就将其定义为成员变量，如果对于整个类都有效，就将其定义为静态变量。

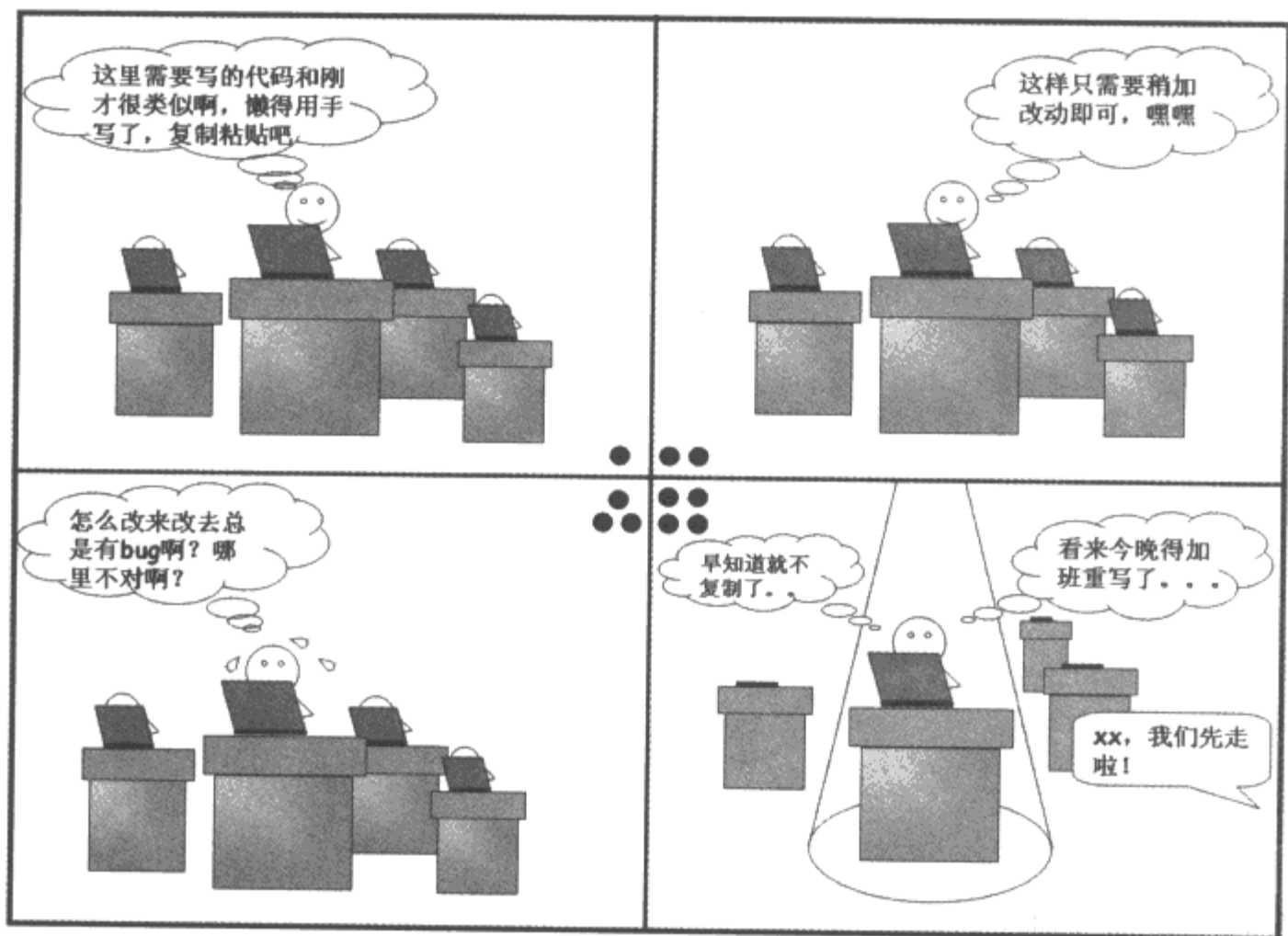


图 10-31 复制再改动带来的弊端

开发人员在设计代码时应该尽最大可能缩短每个变量的生命周期，生命周期越短，所占的系统资源就越少，而且使用起来也很高效。如局部变量存储在内存的栈中，访问速度也相对较快。不要以为这些都是无关紧要的小问题，往往前头费事，后头才会省事。

“师兄，你说得太对了，越是小的问题越要关注，我回去好好修改，争取消灭所有老寿星！”

“不过你可不要一味地将变量的生命周期缩短啊，当一些变量需要长期存在时，我们还是要要有这个气量给他寿命的。”

“啊，比方说呢？”



“比如说你游戏中的那个炸弹 Bomb 类，其中有个表示炸弹大小的数值，你用了个变量而不是魔法数字很值得表扬，但是你这个变量定义为成员变量就不应该了。”

“那定义成什么啊？静态变量？”

“最好是静态变量，因为很多其他类的实现方法中都需要知道 Bomb 对象的大小，甚至在创建 Bomb 对象之前就需要知道，所以定义成静态变量就可以方便其他类及时地使用了。”

“嗯，说得也是啊。”

“总结一下，我们给变量下定义时采取的原则就是量体裁衣，尽量缩短。”

## 10.6 本章小结

本章通过大量实例的列举，将 Java 开发人员在工作中容易落入的圈套和陷阱展现给各位读者朋友。很多时候这些误区的危害并不能在单机程序中体现，如“小肥猪”问题等，但是千万不要因此而忽视它们。这些都是 Java 开发人员通往大牛路上的荆棘和障碍，希望读者朋友在阅读完本章后不会再落入这些圈套和陷阱，以防一招不慎废了自己辛辛苦苦修炼的武功。

# 第 11 章 没有必杀技，怎么敢出来混

在 IT 职场打拼，每个开发人员除了要掌握能让自己生存的基本技能之外，还需要掌握一两种必杀技来出奇制胜。就像初入江湖的小生一样，除了基本招式之外，总要有几个能够制敌的狠招，这样才能够立名于江湖，然后才能够成就武学宗师的“一招致命”。

本章就来为那些还不到“武学宗师”级别的 Java 开发人员传授几种有必要掌握的必杀技，这些技能一般不会用来保命，而是用来将自己的功夫逐步提升，最终达到炉火纯青的境界，使自己的江湖之路走得更加保险。

## 11.1 精通 SQL

没错，SQL 就是第一门要介绍的必杀技。SQL 本书已经在前面的章节中多次提到，其重要性也屡次强调。本节就来对 SQL 做一个简单全面的介绍，看一看 SQL 到底和 Java 区别在哪，为何使用 SQL 会大大提高生产效率，以及如何驾驭好 SQL。

### 11.1.1 掀起 SQL 的盖头来

“师兄啊，我现在非常有危机感啊。”

“为什么呢，你不是在你们公司混得不错嘛。”

“不错是不错，不过我发现和公司其他人相比，我会的他们都会，他们不会的我也不会。我们之间除了经验有别，其他方面基本上都是平起平坐。我缺乏一种能够让自己脱颖而出的技能啊！”

“呵呵，想得还挺长远，能让自己出类拔萃的技能有很多呢，就看你要不要学了。”

“要，要，一定要学啊！师兄，你就看在我是你超级粉丝的份上，不吝赐教吧。”

“好，首先来说一说作为一名 Java 开发人员比较重要的能力——SQL。”

“SQL 我在使用 ORM 时倒是对其有所耳闻，有那么重要吗？师兄你开讲吧。”

SQL 是应用于关系型数据库系统的数据库操作语言，关系型数据库在 1970 年左右提出，目前市面上主流的数据库系统都是关系型数据库。SQL 作为一门语言，和 Java、C/C++ 等通用编程语言有着很大的区别，其主要特性有以下几个方面。

- 非过程化

SQL 是非过程化语言，而 Java、C/C++ 等则是过程化语言。这里的过程化和非过程化不同于面向对象和面向过程的概念，过程化语言需要告诉机器每一步要干什么以及怎么干。开发人员虽然没有直接参与到具体的计算中，但是机器的执行流程开发人员都是了如指掌的。

而非过程化语言只需要告诉机器做什么即可，具体机器是如何做的，开发人员不必关心和过问。非过程化语言的执行机制对于开发人员来说是透明的，比如一句简单的检索语句“select \* from myTable”就只告诉了计算机该干什么，其他的由计算机自己去实现。

SQL 的非过程化特性使其在完成复杂的检索任务时有着不可比拟的优势。非过程化语言的编程就好像是买煎饼果子，只需要告诉老板自己要什么即可。而过程化语言的编程则像是装修房子，虽然有装修工人的工作，但是具体的流程和实现过程必须有房主即开发人员事必躬亲，二者的区别如图 11-1 所示。

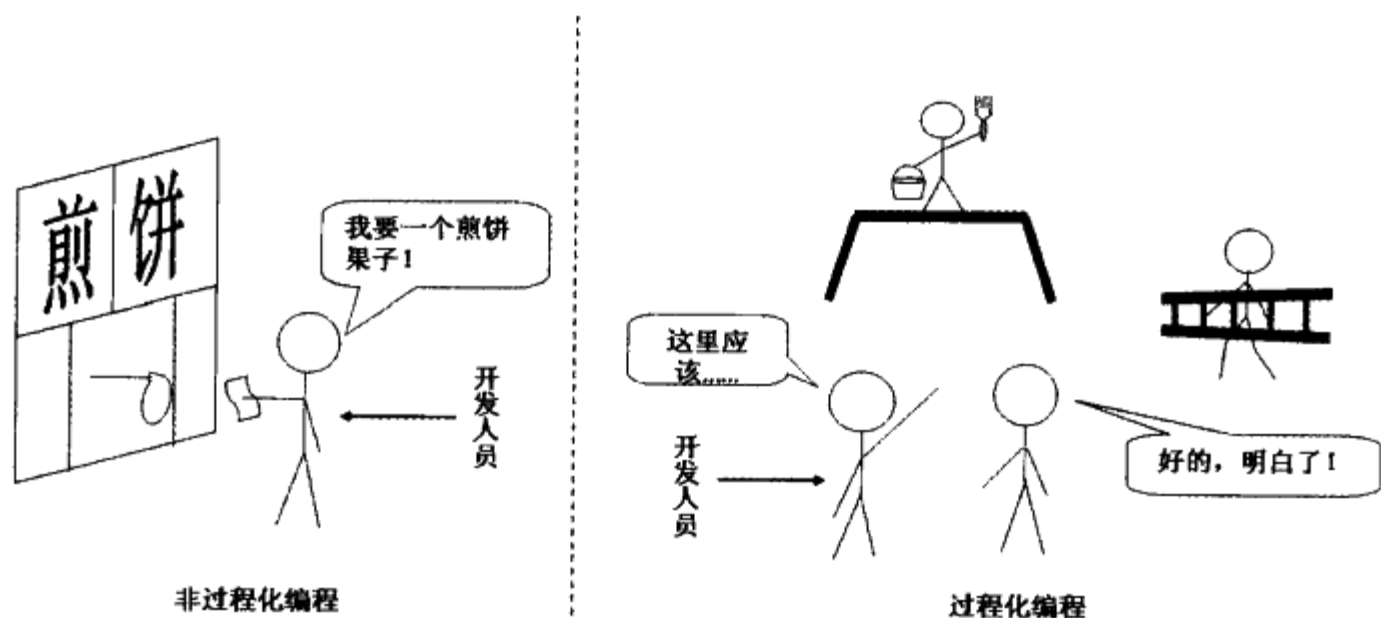


图 11-1 过程化与非过程化编程的对比

#### ● 轻松嵌套

每一条 SQL 语句都有输入和输出，而且输入和输出都是同一种类型的操作对象，叫做记录集（本质上记录集就是一张二维表）。这就允许使一条 SQL 语句接收另一条 SQL 语句作为自己的输入，同时自己输出的记录集也可以作为其他 SQL 语句的输入，这就是嵌套功能。

SQL 嵌套的特性使得用 SQL 可以写出描述复杂功能的简单语句，而这些功能用其他编程语言来写可能会很长很烦琐，因此嵌套使得 SQL 更具有灵活性，功能也更加强大。

#### ● 一家独大

在数据库领域 SQL 已经成为工业标准，是广泛应用的数据库操作语言，市面上主流的商用数据库系统如 Oracle、DB2、MySQL 等都支持 SQL 编程，而且 SQL 的通用性极好，在 Oracle 平台下学习的 SQL 语句，绝大部分也可以应用在 MySQL、DB2 等其他平台上。因此，在数据库领域有这么一句话：“学好 SQL，走遍天下都不怕”。

“师兄，SQL 有这么多的优点，那么具体到我们 Java 开发人员来说，学习 SQL 编程的重要性体现在哪里呢？”

“使用 SQL 可以高效地完成一些复杂的任务，这些都不是可视化的数据库管理工具所能够办到的，不会 SQL 将会使自己的数据库操作能力十分有限。”

“嗯，这是对于数据库管理方面，那么对于软件开发方面呢？”

“在软件开发中遇到对数据库操作的情况时，使用通用的编程语言显然很不方便，而且效率低下，而使用 SQL 就方便多了。”

“的确啊，师兄你这么一说让我想起了 ORM 工具，ORM 不能代替 SQL 的原因就在于此。它做得再好，目前也比不上 SQL 的正宗和高效。”

“最后，掌握 SQL 可以对数据库中的数据进行更好的分析、汇总，这对于满足客户的一些复杂需求很有帮助。”

在目前的学校教育中，对于 SQL 的讲解一般都很浅显，大部分都只是在数据库原理、SQL Server 等课程中客串一把，很少有学校将 SQL 作为一门编程语言来专门讲授。这也是很多开发人员不仅对 SQL 掌握得不多，对其也不够重视的原因之一了。

其实 SQL 非常博大精深，其包涵的内容一点也不比 Java 少。举个例子来说吧，很多人认为自己掌握了 select 语句就很厉害了，再知道个 where 子句就更了不起了。其实单就这一个 select，用最简练的计算机语言表示方式语法树来描述的话，也要洋洋洒洒的 20 页左右才能说明白。很多人都将 SQL 看低了，不过相信在下一小节中，本书将会改变大部分读者的这种想法。

11.1.2 强大的 SQL

前面的小节向读者简单介绍了 SQL 的特性，本小节将为读者展现其在数据库处理中的强大之处，以证明将其作为第一招必杀技的必要性。

“蔡佳娃，听了这么多理论，你是不是对 SQL 的强大功能只有个概念性的理解，有些隔岸观火，未身临其境的感觉啊？”

“还是师兄最了解我，是啊，听了这些道理，还是没有一个切身的体会呢。”

“好吧，我们就来一些真实的体验，让你在 SQL 这个巨人面前发发抖。我来给你出几道跟数据库操作有关的题目，让你见识见识它的威力。”

“好啊好啊，我最喜欢的就是实例了！”

下面将列举几个典型的数据库操作案例。

● 案例一

数据库中有如表 11-1 所示的一张记录实验材料使用情况的表 materialLog，其中 mSequentialID 字段为材料使用的流水号，mClass 为使用的材料类型，mDosage 为材料用量。

表 11-1 materialLog数据库表结构

mSequentialID（材料使用流水号）	mClass（材料类型）	mDosage（材料用量）
111222001	甲	45
111222002	乙	75
111222003	丙	39
111222004	丁	49
111222005	戊	66

有如下的查询需求：列出所有材料的使用记录，结果表中每行包括六个列，分别为材料使用流水号、甲材料量、乙材料量、丙材料量、丁材料量和戊材料量。材料使用流水号对应数据库表中的 mSequentialID 字段，输出时每三位用“-”隔开，甲、乙、丙、丁、戊材料量根据数据库表中 mClass 字段值的不同而改变。最终的输出结果如表 11-2 所示。

表 11-2 查询结果输出表

材料使用流水号	甲材料量	乙材料量	丙材料量	丁材料量	戊材料量
111-222-001	45				
111-222-002		75			
111-222-003			39		

续表

材料使用流水号	甲材料量	乙材料量	丙材料量	丁材料量	戊材料量
111-222-004				49	
111-222-005					66

这个案例中如果不充分使用 SQL 而主要用 Java 来开发的话，可能采取的解决方案可以用如图 11-2 所示的流程图来描述。

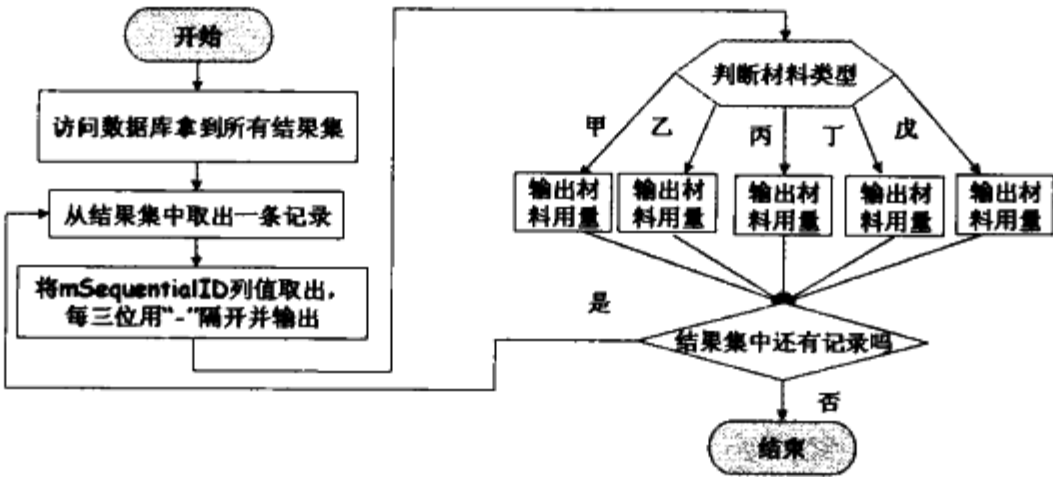


图 11-2 使用 Java 解决问题的流程图

图 11-2 中的流程只在第一步通过最简单的一句“select \* from materialLog”数据库操作得到包含表中所有信息的结果集。然后大量的操作集中在 Java 这边，这样的流程显然过于烦琐，如果数据量很大的话，对系统的性能将有很大影响。

而如果使用一些简单的 SQL 函数，这个问题就可以用非常简短的 SQL 代码来实现，具体代码如下所示。

```
1  select
  substr(mSequentialID,1,3)||'-'||substr(mSequentialID,4,3)||'-'||substr(mSequentialID,7,3) "材料使用编号",
3  decode(mClass,'A',TO_CHAR(mDosage,'99999'),' ') "甲材料量",           //根据材料类型的不同,
4  decode(mClass,'B',TO_CHAR(mDosage,'99999'),' ') "乙材料量",           //输出不同的值
5  decode(mClass,'C',TO_CHAR(mDosage,'99999'),' ') "丙材料量",           //若材料类型匹配,则
6  decode(mClass,'D',TO_CHAR(mDosage,'99999'),' ') "丁材料量",           //输出具体用量
7  decode(mClass,'E',TO_CHAR(mDosage,'99999'),' ') "戊材料量"           //若不匹配则输出空串
8  from material;
```

上述代码中 substr 函数的作用是从指定字段的指定位置截取指定个数的字符；decode 函数的作用是把字段的值按照不同的情况翻译成其他的形式输出，有些类似于 Java 中的 switch 语句；TO\_CHAR 函数将输入转为指定格式的文本。以上代码运行后输出的结果即是如表 11-2 所示的情况。

“怎么样，蔡佳娃，尝到 SQL 的威力了吧？在这个案例中，仅仅是运用了几个 SQL 中的函数，就可以将复杂的业务逻辑变成一句 SQL 语句去实现。如果用 Java 去开发，代码肯定比这个长多了。”

“虽然用 Java 翻来覆去地找要慢，可是你这个 SQL 语句不是也调用了很多 SQL 函数吗？而且那个语句那么长，也不会快到哪去吧？”



“你要知道数据库操作中的耗时多少很大程度上是根据执行 SQL 语句的数量来决定的，SQL 执行引擎在收到 SQL 语句后会自动优化执行，因此这个稍微长一些的 SQL 语句执行的效率也是相当高的。”

● 案例二

数据库中有两张表：author 表记录作者的相关信息，如表 11-3 所示；work\_book 表记录作者所写著作的信息，如表 11-4 所示。要求完成如下更新需求：从 work\_book 表中统计出每位作者所写著作的数量，把统计的数量自动批量更新到 author 表中。

表 11-3 author表数据信息

aid（作者编号）	aname（作者姓名）	acountbooks（作者著书数量）
1001	吴亚峰	
1002	罗贯中	
1003	司马迁	
1004	左丘明	

表 11-4 work\_book表数据信息

aid	wbname	aid	wbname
1001	《Java SE 6.0 编程指南》	1004	《左传》
1002	《三国演义》	1002	《隋唐志传》
1003	《史记》	1004	《国语》
1001	《30 天学通 Java Web 项目案例开发》	1001	《精通 Netbeans——Java 桌面、Web、企业级程序开发》

同样，首先考虑主要用 Java 求解问题，可以画出如图 11-3 所示的流程图。

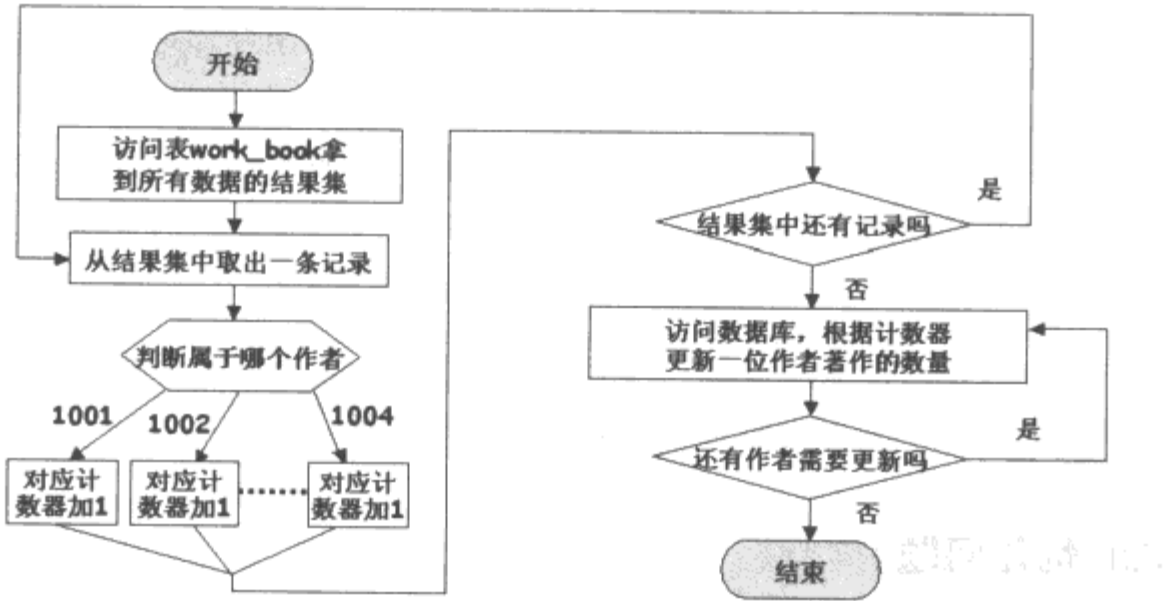


图 11-3 用 Java 解决问题的流程图

从图 11-3 中可以看出，将 work\_book 表中的数据结果集拿到后，对每条记录进行遍历，是哪个作者的著作，就将后台维护的对应作者计数器（可以用数组实现）加 1。遍历结束后会获得每位作者的著作数量，然后分别对 author 表中每位作者做一次数据库的更新操作，将著作数量更新到数据库中。

当然也可以在遍历 work\_book 表时每得到一条记录，就对 author 表执行一次更新。这样省去了后台计数器的开销，但是过于频繁地访问数据库也会降低性能。无论怎样优化，用 Java 解决这个问题都很难得到比较满意的执行效率。

下面就该介绍最王道的解法了——使用 SQL，这项繁重的劳动只用一句 SQL 语句就可以完成，代码如下所示。


```

1      update author aa                                //最外层是更新操作的 SQL 语句
2      set accountbooks=                                //设置作者著书数量
3      (
4          select bs from                                //内层嵌套了将对应作者著书数量检索出来的 SQL 语句
5          (
6              select count(*) bs,aid naid from work_book wa
                                                         //最内层是从 work_book 表中
7              group by wa.aid //统计每个作者数量的 SQL 语句
8              union
9              select 0,ab.aid from author ab //work_book 表中不存在的作者
10             where ab.aid not in //其著书数量统计为 0
11             (
12                 select wc.aid from work_book wc
                                                         //检索出 work_book 表中
13                 group by wc.aid //作者编号的集合
14             )
15         )
16     where naid=aa.aid                                //内外层的 SQL 语句条件挂接
17 );
```

以上代码虽然比较长，但是执行的时候却是将其作为一句嵌套检索语句执行的，而用 Java 开发的话，不仅代码更加冗长而且效率也很低下。如果将二者放到服务器上进行对比，从客户端的体验来看，性能不知道要差多少倍，由此可见 SQL 在处理数据尤其是处理复杂数据操作时的绝对优势。

“师兄，我终于见识到 SQL 的强大，以前我对于 SQL 的学习都是管中窥豹，只见一斑。总觉得 SQL 只是 Java 开发人员手中的一个辅助工具，仅仅是用来打开数据库大门的钥匙而已。”

“不光是你这样，很多开发人员都没有对 SQL 引起足够的重视。有这类思想的开发人员在遇到客户复杂的数据处理需求时，会因为过分使用 Java 而使得开发过程痛苦万分。同时客户用起来也会因为性能问题头痛连连，这种‘双输’的局面才真是悲剧呀。”

 **提示** 要读懂本小节展示的 SQL 代码，需要有一些 SQL 的基础，本书由于篇幅有限将不再赘述，有兴趣的读者可以自己去看其他资料和书籍学习。

### 11.1.3 SQL 优化问题

“师兄，听你这么一说，SQL 果然很神奇啊！你刚才说的那些 SQL 语句有的我都不太明白，看来学好 SQL 真的可以让自己不知不觉地胜人一筹呢。”

“是啊，不过要想成为真正的 SQL 大牛，光会这些嵌套检索的高级语句还不够呢。”

“哦，还需要掌握什么啊？”

“在掌握高级语法的基础上，应该再熟悉一些 SQL 的语句优化方面的知识，这样才可以真正走遍天下都不怕呢。”

在软件系统的适当位置，用短小精悍的 SQL 语句替代繁琐 Java 语言，可以很好地优化系统

的性能，但是 SQL 语句本身需不需要优化呢？答案是肯定的。通过优化 SQL，小小的变动也可以大大地提高系统性能。下面就介绍几个经常用到的优化方法，以抛砖引玉。

● TOP-N 问题

在进行检索时经常会出现这样的一类需求：只提取检索结果中的前  $N$  条记录，或者结果中从第  $M$  条到第  $N$  条的记录。如查找学生总成绩的前 100 名，或是查找收入排名 10~20 位的员工等。


这种情况下如果用 Java 对提取出来的数据结果集进行二次操作会显得很累赘，而如果将嵌套检索和 ROWNUM 伪列结合起来使用，就可以轻松达到这种效果了。下面给出一个示例，该示例从学生表中查询年龄排序为 4~6 位的学生，并将其信息列出，学生信息表如表 11-5 所示。

表 11-5 student表数据信息

sid (学号)	sname (学生姓名)	sage (学生年龄)
1001	Tom	27
1002	Jerry	26
.....	.....	.....
1007	Jane	23
1008	Michael	25

配合使用嵌套检索和 ROWNUM 伪列的 SQL 代码如下所示。

```
1  select sid,sname,sage from                                //最外层检索出 4~6 名的学生信息
2      (
3          select rownum temp_no,sid,sname,sage from        //中间层检索通过伪列分配编号
4              (
5                  select sid,sname,sage from student        //最内层检索学生的信息并
6                  order by sid                               //根据年龄排序
7              )
8      )
9  where temp_no between 4 and 6;
```

 **提示** 关于 TOP-N 功能的实现，不同的数据库产品虽然都支持，但是在语法上也存在些许的差异，上述代码是以 Oracle 环境下的 TOP-N 实现为例进行说明的。

● 恰当使用索引

数据库中，一个简单索引是由指定字段值和 ROWID 组成的一个隐藏子表。ROWID 相当于平时书目录当中的页码，可以通过 ROWID 快速找到记录存储的物理位置。

在数据库操作中，当 select 语句中查找的字段有索引时，索引可以加快查找速度；当 insert 语句中有索引时，由于系统需要维护索引，所以会降低 SQL 语句的执行速度；而在 delete 和 update 语句中出现索引时，则加快和降低语句执行速度的可能性都存在。因此在使用索引时需要注意，避免不正当的使用引起系统执行效率的降低。

● 用 case 表达式替代多次查询

当需要对一个表中的同一条记录执行多种运算时，使用 case 表达式而不是多次查询会提高系统的执行效率。下面给出一个使用 case 表达式的例子，本例也是基于表 11-5 来进行操作的，要求将表中 25 岁以下、25 到 30 岁、30 岁以上的学生分别输出。

如果采用多次查找，其代码如下所示。

```

1  select count(*) "25岁及以下" from student where sage<=25;
2  select count(*) "25岁以上到30岁" from student where sage>25 and sage<30;
3  select count(*) "30岁以上" from student where sage>=30;

```

而如果使用 case 表达式的话, 会将三个 SQL 语句整合为一个带有 case 表达式的查询语句, 代码如下所示。

```

1  select
2      count(case when sage<=25 then 1 else null end) "25岁及以下",
3      count(case when sage>25 and sage<30 then 1 else null end) "25岁以上到30岁",
4      count(case when sage>30 then 1 else null end) "30岁以上"
5  from student;

```

#### ● 用 where 子句代替 having 子句

where 子句用于过滤行, 而 having 子句用于过滤组。having 子句的过滤需要先将行进行分组, 这会消耗掉系统一定的 CPU 时间。因此为了提高速度, 应该尽量使用 where 子句, 这样就避免了对不需要的行进行分组, 下面两段代码就说明了这个问题。

用 having 子句进行过滤, 代码如下所示。

```

1  select fielda,ave(fieldb)from table1
2  group by fielda
3  having fielda=1;

```

用 where 子句进行过滤, 代码如下所示。

```

1  select fielda,ave(fieldb) from table1
2  where fielda=1
3  group by fielda;

```


从上面两段代码中可以看出, 第二段代码先用 where 子句过滤了不必要的行, 再用 group by 子句分组, 这样可以在一定程度上提高执行效率。

#### ● 多表连接时要注意的问题

在实际开发中, 很多时候需要进行表的连接查询, 这个时候出于系统性能考虑需要注意的问题有以下两点。

第一就是必须选择正确的连接顺序, 将行较少的表放到后面连接, 例如要连接三个表 table1、table2、table3, 假设其行数分别为 1000、10、100, 则连接顺序应为先将 table2 连接到 table3 上, 再将 table3 连接到 table1 上。

第二个需要注意的要点就是尽量使用全称列名, 全称列名一般只用在相连接的几个表中有同名列的场合。但是考虑到系统效率的问题, 在连接的时候对所有的列应该都使用全称列名, 这样系统就不会在另外花时间自动决定某一列属于哪个表了。

 **提示** SQL 方面的必杀技其实三天三夜都说不完, 本书只是介绍了其中极少的一部分以引起读者对这个方面的重视, 需要的读者应该参照专门的教程或书籍单独学习。

### 11.1.4 当下主流的数据库产品

“师兄, 我觉得光学学嵌套检索之类的 SQL, 不用研究什么优化, 也够我用了。”

“看看, 是不是又想给自己放低标准啊, 休想有这个念头。况且有关 SQL 的知识我还没说完呢。”

“啊，那还有什么啊？”

“SQL 只是一门语言，实际开发中都是将其运用到特定的数据库中，因此市面上的主流数据库产品当然也就必须要了解了。”

SQL 作为数据库领域的通用语言和一种工业标准，自然也成就了很多数据库产品。见识了 SQL 的博大精深之后，我们就来了解一下市面上流行的几款数据库产品吧。

### 1. DB2

DB2 的 logo 如图 11-4 所示，DB2 是 IBM 公司的关系型数据库管理系统。作为 IT 史上的蓝色巨人，IBM 多次走在世界的前沿，最早关系型数据库管理的思想就诞生在 IBM。而 SQL 也是由 IBM 开发后被纳为工业标准的，而 DB2 则被誉为最早使用 SQL 的数据库产品。

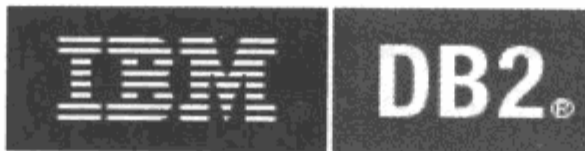


图 11-4 DB2 的 logo

DB2 可以有从手持式设备到企业级服务器的多种版本的实现，同时也提供不同系统平台下的不同版本。除了命令行之外，DB2 还提供了一个 Java 客户端。DB2 为包括 Java 在内的多种编程语言提供了数据库操作的 API，其主要应用在银行、医疗等领域。

### 2. Oracle

Oracle 数据库产品的 logo 如图 11-5 所示，由甲骨文公司推出。甲骨文公司是世界上最大的数据库软件公司，其名下的数据库服务器也是 IBM 最强力的竞争对手。



图 11-5 Oracle 的 logo

Oracle 数据库产品在很多技术上都是全球领先，如在集群中提供互联网数据服务等，其最大的特点就是可以和配套的 Oracle 应用服务器、Oracle 开发工具套件方便地集成为一个电子商务平台。Oracle 目前主要应用在电信业和一些大型企业中。

### 3. Microsoft SQL Server

Microsoft SQL Server 的 logo 如图 11-6 所示，它是微软在关系型数据库方面的解决方案，SQL Server 最初由微软和 Sybase 等公司共同研发，后来中止与上述公司的合作后，微软主要关注于其在 Windows 平台下的应用。



图 11-6 SQL Server 的 logo

SQL Server 的最新版本为 SQL Server 2008，这个版本是其发展历程中一个比较重要的版本。应对于新技术的发展，添加了许多新特性，如对 .NET 平台的支持等，SQL Server 目前主要作为中小企业的数据库管理解决方案。



#### 4. MySQL

MySQL 的 logo 如图 11-7 所示, MySQL 最初由一家瑞典公司推出, 后被 Sun 收购。2009 年 Sun 又被 Oracle 收购, 于是 MySQL 的去向也颇受人关注。不过 Oracle 已经在公开场合承诺不会放弃 MySQL, 因此使用 MySQL 的读者也不用担心。



图 11-7 MySQL 的 logo

MySQL 是开放源码的数据库系统, 与诸如 Oracle、DB2 等大型数据库产品相比, MySQL 在数据处理规模和功能上相对都有些欠缺。但是它具有简便、轻巧、免费等特性, 互联网上较流行的一种网站架构方式 LAMP (Linux、Apache、MySQL、PHP) 就是采用的 MySQL 数据库。MySQL 多用在一些开源社区或论坛网站, 如维基百科后台就有一些部分使用了 MySQL。

目前也有一些银行的二级数据库开始采用 MySQL, 同时 MySQL 也逐渐受到了很多中小企业的青睐, 相信在未来应该会发展得更好。

#### 5. Derby

Derby 的 logo 如图 11-8 所示, Derby 不仅是开源的, 而且是纯 Java 开发的关系数据库。Derby 的前身是 IBM 开发的 CloudScape 数据库, 后来 IBM 公司将其捐献给 Apache 软件基金会, 随后 Sun 也为其捐献了一个开发团队。



图 11-8 Derby 的 logo

从 Java SE 6.0 开始, Java SE 中就自带了 Derby, 不像其他的数据库产品, Derby 不用安装即可直接使用, 虽然小巧, 但是 Derby 的功能并不弱, 最高可以管理几十 GB 的数据。同时由于是纯 Java 开发, Derby 也具有天生的跨平台性。

Derby 除了可以像其他数据库产品在 C/S 架构下服务, 也可以工作在嵌入式模式下, 与 Java 程序运行在同一个虚拟机上。这样就免去了其他数据库系统需要做的数据源配置等问题, 使用起来非常方便。Derby 的这种特性可以作为测试平台, 即先将程序运行在 Derby 数据库平台, 测试无误后再移植到其他平台。

## 11.2 拿下正则式

不管是从事哪种类型的软件开发, 很多时候都离不开对字符串的操作, Java 开发中也是如此。而如果想要轻松灵活地驾驭字符串, 拿下正则式将是必须要完成的任务。正则式是处理和分析字符串时最有效的武器, 本节将重点讲解正则式的语法规则和特性, 然后将分别在 Java 和 JavaScript 中运用正则式的思想去解决一些实际问题。

### 11.2.1 细说正则式

“师兄，上学的时候我的一个老师说字符串是最灵活的一种数据类型，很多实际应用中都是将其他类型转变为字符串再进行相应操作的，现在想想，的确是这样啊！”

“不错，字符串处理起来是比较方便。”

“我发现 Java 中为字符串对象提供的那些方法太有用了，要是没有它们，估计现在要对字符串进行处理时还得去研究数组之类的麻烦东西呢。”

“呵呵，这你就满足了啊，看来你遇到的字符串处理问题还是太少。操作字符串真正的强力工具应该是正则式呢，学会这个你才算是搞定了所有的字符串问题。”

“啊，正则式有这么厉害吗？”

通常遇到的字符串，虽然是内容各异、长短不一，但是很多情况下，一些字符串还是遵循一定的规律和模式的，比如表示日期的字符串、表示电话号码的字符串等。而正则式就是携带这种模式信息的一种特殊的字符串。使用正则式可以很方便地对字符串进行匹配查找、替换和分割等操作。

例如需要对一段文本进行处理，将文本中出现的诸如“4:15pm”、“11:20pm”等改为“16:15”、“23:20”这样的 24 小时制表示法，如果用 Java 直接开发的话，肯定是可以的。因为正如蔡佳娃说的那样，String 对象提供了很多方便实用的字符串操作方法。在替换之前首先要在漫漫文本中找到符合格式的子串，可以设想一下这个工作流程，如图 11-9 所示。

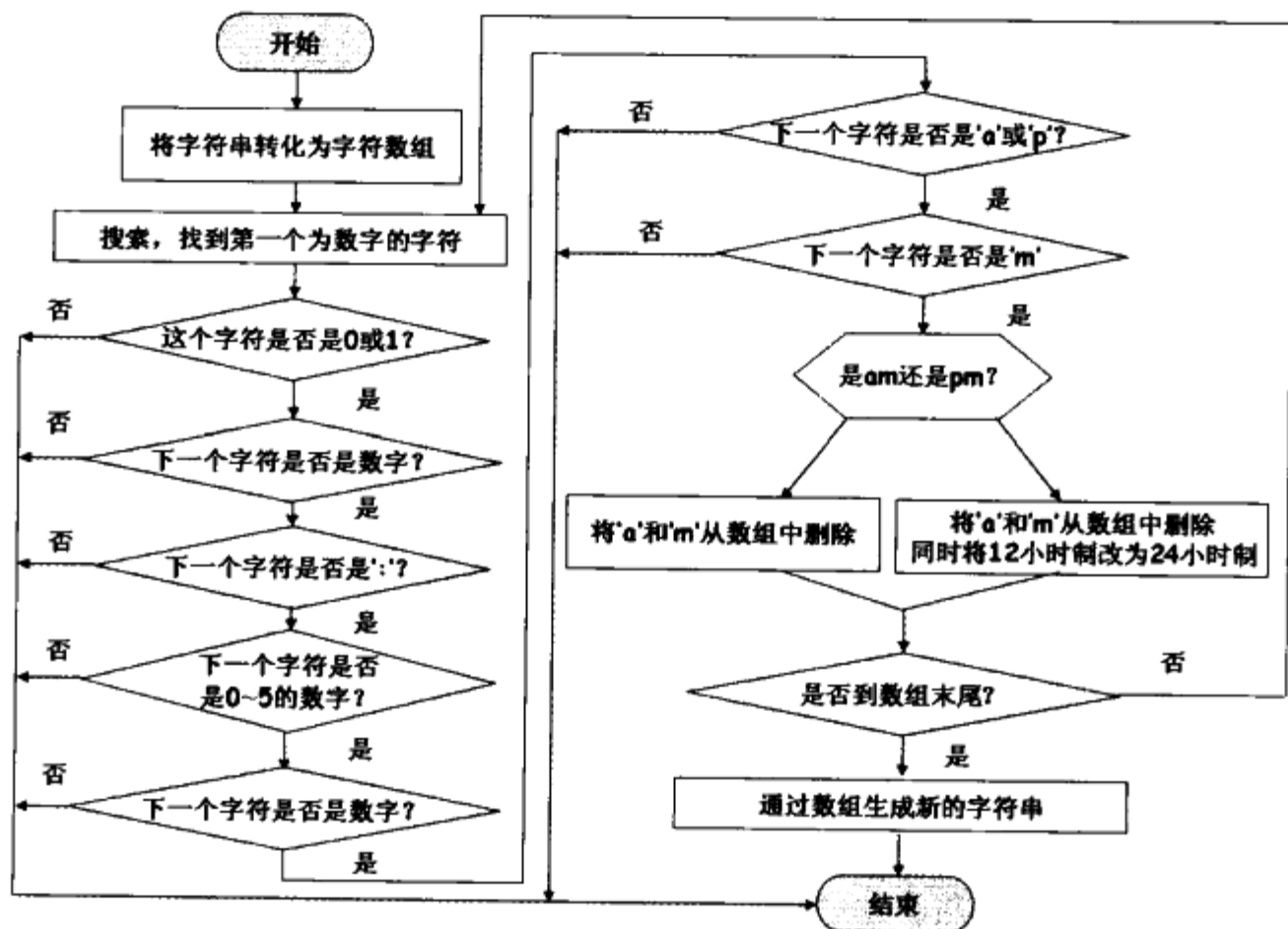


图 11-9 不使用正则式的工作流程

当然可以用 StringBuffer 来替代数组简化开发，但是仍然无法改变其编程烦琐和效率低下的命运，而且这种开发还很有可能出现 bug。如 08 和 8 的问题，自己编程解决就很麻烦。而如果使用正则式的话，只需要下面一句话就可以搞定。

```
1 (0?[1-9]|1[0-2]):[0-5]\d[a|p]m
```

以上正则式将匹配按 12 小时制表示时间的字符串，如“09:53am”、“8:54pm”等。这种方式不但快捷高效，而且出错率低，由此可见正则式在字符串处理方面的强大优势。

一个正则式字符串可能第一眼很难看懂，因为在正则式中每个字符都代表着不同的含义。正则式中各个字符的含义及组合规则构成了正则式的语法，常见的语法要素说明如下。

1. 普通字符

如字母、汉字、数字、下画线等，这些字符在正则式中只匹配该字符本身。如正则式“a”所匹配的就是所有只含字母“a”的字符串。

2. 预定义字符

可以表示某个范围内的任意一个字符，常见的预定义字符及其含义如表 11-6 所示。

表 11-6 预定义字符对照表

预定义字符	说 明
\d 和 \D	\d 表示 0 到 9 之间的一个数字，而 \D 则是 \d 的补集，表示 0 到 9 以外的任意一个字符
\w 和 \W	\w 表示字母、数字、下画线中任意的一个字符，而 \W 则表示除字母、数字、下画线之外的任意一个字符
\s 和 \S	\s 表示任意一个空白字符，如空格、制表符、换页符等，而 \S 表示空白字符以外的任意一个字符

运用预定义字符和普通字符可以组成一些简单的正则表达式，请看下面的一个正则式。

```
1 010-8\d\d\d\d\d\d\d
```

上述正则式可以用来匹配区号为 010、首位为“8”的电话号码。

3. 自定义字符组

有时预定义字符并不能完全满足需要，这时候可以用自定义字符组来表示。自定义字符组是包含在“[”和“]”之间的一系列字符，它表示所包含字符中的任意一个字符，如[wyf]表示 w、y、f 之间的任意一个字符。

表示自定义字符组经常用到“-”，如[a-z]表示 26 个小写字母中任意一个，[2-8]表示 2 到 8 之间的一个字符。“-”表示的是 Unicode 介于两者之间的任意字符。使用“-”需要注意其次序，如[-az]和[az-]表示的均是“a”、“z”、“-”三者中的任意一个字符，而不是某个范围。

与“[”和“]”相对的是“^”和“|”，其表示自定义字符组之外的任意一个字符，如[^a-zA-z]表示非英文字母的字符。“^”除表示补集之外，还是一种边界符。

继续完善刚才提到的匹配电话号码的正则式，如果需要匹配号码首位在 5~8 之间的电话号码，则可以用如下正则式表示。

```
1 010-[5-8]\d\d\d\d\d\d\d
```

4. 边界符

边界符在进行模式匹配时不与任何字符匹配，只是代表某个分界点。正则式中的边界符主要有字符串起始分界点和单词边界两种。

字符串起始分界点由“^”和“&”表示，分别代表字符串的开始和结束位置，通常用“^”和

“&”来说明正则式要匹配的是整个字符串的内容，而不是其某个子串。如“`^[01]\S&`”表示由“0”或“1”开头、后跟一个空白符之外的任一字符。字符串“1t”可以和其匹配，但“w1tb”则不能。

单词边界用“`\b`”来表示，其表明正则式要匹配的内容是整个单词，而不是单词的一部分或多个单词。例如“`\bJack\b`”与字符串“Jack Jackson”进行匹配时，只会和“Jack”进行匹配，而不会和“Jackson”匹配。“`\B`”表示非单词边界，即“`[^\b]`”，使用“`\b`”可以方便地提取某段英文中的单词。

“哦，原来就是这样啊，正则式也不难嘛，我回去多写几个例子就基本掌握了。”

“看看，又不虚心了吧，这才只是正则式中一些基本的语法，很多时候只有这些基本写不出什么高效实用的正则表达式。这些简单的你能听懂是应该的，后面的我就不敢保证了。”


“啊，那师兄你接着讲吧，我一定仔细听。”

5. 量词

正则表达式中运用得最多的是量词，量词表示某个字符或组重复在字符串中出现的次数，正则式匹配中经常用到的量词如表 11-7 所示。

表 11-7 正则式量词表

量 词	说 明
S?	表示指定的字符或组最多出现一次，可以没有
S+	表示指定的字符或组至少出现一次，多了不限
S*	表示指定的字符或组可以出现任意次，包括不出现
{n}	表示指定的字符或组必须出现 n 次
{m,n}	表示指定的字符或组最少出现 m 次，最多出现 n 次
{n,}	表示指定的字符或组最少出现 n 次，多了不限

 **提示** 表 11-7 中的 S 不仅表示单个字符，还表示字符组，字符组的概念马上会介绍到。

量词的使用可以大大提高正则式的灵活性，如“`\d{2}[a-zA-Z]*`”表示以两个数字开头、后跟可以为空的任意字母序列的字符串。另外，刚刚匹配电话号码的正则式也可以简化为如下所示。

```
1 010-[5-8](\d){7}
```

6. 其他字符

正则式中还有许多其他字符表示特殊的含义，这些字符的作用如表 11-8 所示。

表 11-8 正则式特殊字符表

特殊字符	说 明
<code>\r、\n、\t、\f</code>	依次分别表示回车符、换行符、制表符、分页符
<code>.</code>	表示除换行符以外的任意一个字符，即“ <code>^[^\n]</code> ”
<code> </code>	用来连接两个表达式，表示或的关系
<code>()</code>	包含在括号中的部分将作为一个分组，这样可以方便对多个字符进行整体操作
<code>\n</code>	n 表示任意数字，通常和分组一起用，表示对某个分组的引用
<code>\x</code>	转义字符，x 表示字符
<code>?</code>	只能用在 <code>?</code> 、 <code>+</code> 、 <code>*</code> 、 <code>{m,n}</code> 、 <code>{n,}</code> 后面，用来指明匹配模式为非贪婪模式

下面对转义字符和分组的概念进行简单说明。

#### ● 转义字符

很多符号在正则式中代表一定的含义，当需要表示这些字符本身时，需要在这些字符前面加上转义字符“\”，如需要表示小数点“.”时，应该写为“\.”。类似的需要转义的字符还有“^”、“\$”、“|”、“?”等，需要表示“\”时应该写成“\\”。

#### ● 分组

在正则式匹配中，有时将一些字符看做整体来对待会更有效率，这时候就将它们作为一个组。如需要匹配由任意个交替出现的数字和字母组成的字符串时，就可以将正则表达式写为：“(\d[a-zA-Z])+”，需要分组的部分用“(”和“)”包围起来。

当一个正则式有多个分组时，可以用“\n”来表示对分组的引用，如正则式“((ab)(cd))”分为3组，分别为“abcd”、“ab”、“cd”，分组编号依次为1、2、3。如此一来，正则式“((ab)(cd))\1”就表示对字符串“abcdabcd”进行匹配，“((ab)(cd))\2”表示对“abcdab”进行匹配。也就是说“((ab)(cd))\1”相当于“((ab)(cd))(abcd)”，“((ab)(cd))\2”相当于“((ab)(cd))(ab)”，有了引用技术可以减少代码量。

分组的另一个重要作用是用在匹配成功后访问与该分组匹配的字符串子串。如“(\d[a-z])abcd”匹配字符串“8nabcd”成功，则可以在程序中通过引用的索引得到字符串“8n”。

 **提示** 关于匹配模式的贪婪与非贪婪问题，本书将在后面小节中进行介绍。

“师兄，你说得果然没错，正则式看来的确不简单啊。不过光听你说了半天，印象不深刻啊！”

“呵呵，忘性还真快，我来讲几个应用你就印象深刻了。先回顾最开始我们提到的匹配字符串，你能看明白它吗？”

```
1 (0?[1-9]|1[0-2]):[0-5]\d[a|p]m
```

“哇，刚才都是分着讲的，这下全整合到一起了，还真是不好看哪。”

“看看，功力未到吧，这个我来说一下。”

- 冒号前面的字符表示12小时制的小时，“0?[1-9]”表示“01”到“09”或“1”到“9”，“1[0-2]”表示10点到12点。
- 冒号后面的“[0-5]\d”表示“00”到“60”秒。
- “[a|p]m”表示“am”或“pm”。

“果然啊，听师兄这么一点拨，豁然开朗啊！”

“呵呵，不过光看懂可没用啊，学会了正则式也不是看着玩的，要学着去开发自己的正则式将其运用到实际的字符串处理中才行啊！”

“好的，师兄！我一定向这方面努力！”

正则式的优势几乎涵盖了对于字符串进行操作的所有方面，下面将举例说明。

#### ● 字符串匹配

在开发中许多地方都需要对某个字符串进行模式匹配，如检查密码强度、检查用户名是否符合规范、验证电子邮箱是否正确等。正则式可以帮助开发人员完成看似繁重的劳动，以电子邮件



地址的匹配为例，一个能够匹配正确邮箱地址的正则式如下所示。

```
1 \w+(\.\w+)*@(\w.)*\w+
```

#### ● 字符串查找替换

现实中不仅需要符合某种模式的字符串查找出来，还需要将其进行部分或整体上的替换，如在金融领域经常会遇到将输入字符串转化为数值以进行计算的场合，如“¥899.08”，不仅是 Java，其他任何语言中都没有这种转换，但是如果使用一个正则式就可以轻松解决问题。

```
1 ¥[1-9]\d*\.\d*
```

只需要将目标字符串与正则式进行匹配验证，再将匹配成功后的字符串替换成为可以转化为数值的样式如“899.08”，最后再使用 Java 的一些方法将其转化为对应数值即可。

#### ● 字符串分割

字符串除了可能被替换，还有可能被分割，这里的分割不是简单地按照固定的分隔符进行分割，而是根据匹配某个模式的字符串进行分割。如一个字符串中穿插有不同的日期，但是格式是固定的，例如一个表示日期的正则式如下所示。

```
1 (\\d{4})年([1-9]|1[0-2])月([1-9]|1[0-2]|3[0-1])日
```

如果想以日期作为分隔符切割字符串的话，只需要使用上述正则式进行匹配验证，以字符串中满足条件的地方为界限对字符串进行切割即可。

## 11.2.2 正则式在 Java 中的运用

正则式是一种工具，它需要依附在编程语言上来实现对字符串的处理功能。上一小节本书讲述了正则式的优点和基本语法，这一小节将介绍正则式在 Java 开发中的运用。

“蔡佳娃，刚才谈了谈正则式到底是个什么东西，不知道你听明白了没有呢？”

“明白倒是明白了，具体怎么用还不太了解。”

“好啊，下面我们就来看看在 Java 中是如何运用正则式这个强有力的武器的。”

Java 中提供了两个类用来实现正则式的操作，分别是 `Pattern` 类和 `Matcher` 类，两者均位于 `java.util.regex` 包下。

`Pattern` 类对象表示一个通过编译的正则式，它提供了一些方法进行字符串的匹配和分割等操作，但是不够全面。`Pattern` 对象在创建时可以为它制定字符串匹配的模式，如是否区分大小写、是否启动多行查找模式、是否启动规范等价等。

`Matcher` 类对象表示需要进行模式匹配的字符串或字符序列。`Matcher` 类对象无法通过调用构造器获得，需要通过 `Pattern` 类的 `matcher` 方法获得。`Matcher` 类中提供了进行字符串匹配替换等一系列方法，可以完成很复杂的字符串操作。

下面将使用 Java 中的 `Pattern` 类和 `Matcher` 类实现一个通过正则式对字符串进行字符串匹配和替换的案例，该案例功能描述如下。

一个字符串序列中的英文字母（只含大写）被加密了，所有的字母都变成了形如“@*n*”的密文，其中 *n* 为 1 到 26 之间的整数，表示 26 个大写字母，如“@25”表示“Y”，“@2”表示“B”。

现需要将这段密文还原成明文输出。

根据题目要求，定义如下正则式。

```
1  "@([3-9]|1\d?|2[0-6]?)"
```

- “[3-9]”表示3到9的数值，它代表了由“C”到“J”之间的字母。
- “1\d?”表示1或10到19之间的数值，代表字母“A”以及“K”到“S”之间的字母。
- “2[0-6]?”表示2或者20到26之间的数值，代表“B”和“T”到“Z”之间的字母。
- “[3-9]|1\d?|2[0-6]?”包含在括号中表示分组，便于代码中获得匹配成功的字符串。

实现上述功能的代码如下所示。

```
1  package wyf;
2  import java.util.regex.*;          //引入相关包
3  public class RegDecode{
4      public static void main(String args[]){
5          String patternStr = "@([3-9]|1\d?|2[0-6]?)" ;    //定义正则式字符串
6          String s = "今天是@19@21@14@4@1@25, 天气非常@7@15@15@4。//密文
7              大家@2@25@2, 我先走了。";
8          char [] table={'A','B','C','D','E','F','G','H','I','J','K','L','M','N',
9              //密文解析表
10             'O','P','Q','R','S','T','U','V','W','X','Y','Z'};
11          Pattern pattern = Pattern.compile(patternStr);    //创建 Pattern 对象
12          Matcher matcher = pattern.matcher(s);    //创建 Matcher 对象
13          StringBuffer result = new StringBuffer(); //创建缓冲区用于存放替换后的新串
14          while(matcher.find()){    //找到匹配位置
15              String sIndex = matcher.group(1);    //获取匹配成功后指定分组的字符串子串
16              int index = Integer.parseInt(sIndex); //通过密文求出明文
17              String rep = ""+table[index-1];
18              matcher.appendReplacement(result,rep); //用明文替代密文
19          }
20          matcher.appendTail(result);    //将替换后的内容送入 result 中
21          System.out.println("密文为: \n"+s);    //打印输出密文
22          System.out.println("转换后的明文为: \n"+result.toString()); //打印输出明文
23      }
24  }
```

细心的读者可能会发现，代码第5行定义的正则表达式和之前设计的正则表达式不同。这是因为在Java语言中，“\”也是转义字符，需要用“\\”来表示“\”，因此表示任意一个整数的“\d”在Java代码中则是“\\d”，系统会在编译正则式的过程中将“\\d”转化为“\d”。

上述代码执行之后其结果如图11-10所示。输入的密文“今天是@19@21@14@4@1@25，天气非常@7@15@15@4。大家@2@25@2，我先走了。”，将通过正则式的匹配和替换，输出相应明文：“今天是SUNDAY，天气非常GOOD，大家BYE，我先走了。”。

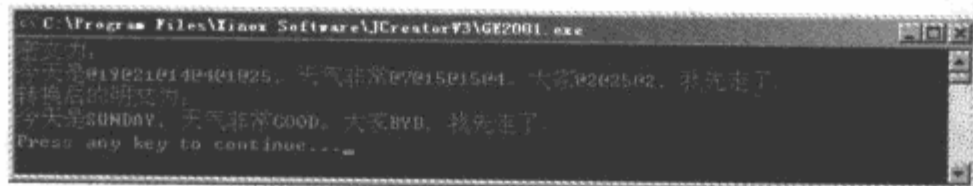


图 11-10 执行字符串替换结果图

“哇，师兄，Java 中提供的类库再加上正则式本身的强大，真是无敌了。”

“呵呵，其实很多时候都用不着强大的 Pattern 和 Matcher 类出马，String 类自身也有很多支持正则式操作的 API 呢。一些小问题用 String 自带的 API 就够了。”

“是吗？这些我怎么都没注意到呢？”

“你总是刚明白个开头就甩头走人，当然不会知道了。”

“那师兄你赶紧帮我把头扭回来吧！”

Java 中 String 对象提供的方法相信很多读者都曾经用到过，如最常见的 replaceAll 方法，但是很多读者对这些方法仅仅停留在非常简单的使用上。如 replaceAll 方法仅仅是将指定的字符串替换成其他字符串。其实这些方法加上正则式的辅助，其能力将会大大提升。

String 对象的方法中使用正则式也可以实现字符串模式匹配的检查、替换等功能。下面给出一个 String 对象使用正则式对复杂字符串进行分割的操作，案例功能描述如下。

一个字符串中记录了两人的聊天记录，每句话后面都会紧跟着形如“-2009 年 9 月 25 日”的日期。现需要将两个人之间用日期隔开的谈话内容提取出来。

本案例中需要开发一个能够匹配案例中表示日期字符串的正则式，该正则式如下所示。

```
1  -(\d{4})年([1-9]|1[0-2])月([1-9]|1[2-9]|3[0-1])日
```

- “\d{4}”表示年份，“[1-9]|1[0-2]”分别表示 1 到 9 月和 10 到 12 月。
- “[1-9]|1[2-9]|3[0-1]”分别表示 1 到 9 日、10 到 29 日、30 和 31 日，中间用“|”分开，表示或关系。
- 正则式中将表示年、月、日的部分进行了分组，虽然本案例中不会使用分组索引，但是这对于以后功能的扩展还是很有好处的。

本案例中代码如下所示。

```
1  package wyf;
2  public class RegDate{
3      public static void main(String args[]){
4          String patternStr="-\\d{4}年([1-9]|1[0-2])月([1-9]|1[2-9]|3[0-1])日";
5                                     //定义正则式字符串
6          String s = "你好吗？-2008 年 8 月 3 日我不好！-2008 年 12 月 25 日为什么？-"+
7          "2008 年 12 月 28 日不知道-2009 年 6 月 23 日"; //定义要匹配处理的字符串
8          //字符串分割
9          System.out.println("原字符串为：\n"+s+"\n");
10         String [] sa = s.split(patternStr,0); //按照正则式的模式去分割字符串
11         System.out.println("分割后的字符串数组长度为："+sa.length); //打印输出长度
12         System.out.println("分割后每个聊天记录为：");
13         for(String tempS:sa){ //打印输出提取的聊天内容
14             System.out.println(tempS);
15         }
16     }
```

上述代码中的第 9 行为“String [] sa = s.split (patternStr,0);”其中 split 方法的第二个参数表示分割限制，可以取正数、负数和零三种情况。参数为正数时 n 表示最大切割份数为 n-1；为负数

则尽可能多地分割；为零表示尽可能地切割，如果末尾遇到空串则舍弃。

实现本案例功能的正则表达式并不唯一，只要能满足条件即可，本例中的正则式也有需要改进的地方。如该正则式将匹配“0000 年 8 月 24 日”成功，这显然是不符合实际的。本例中代码的运行结果如图 11-11 所示。

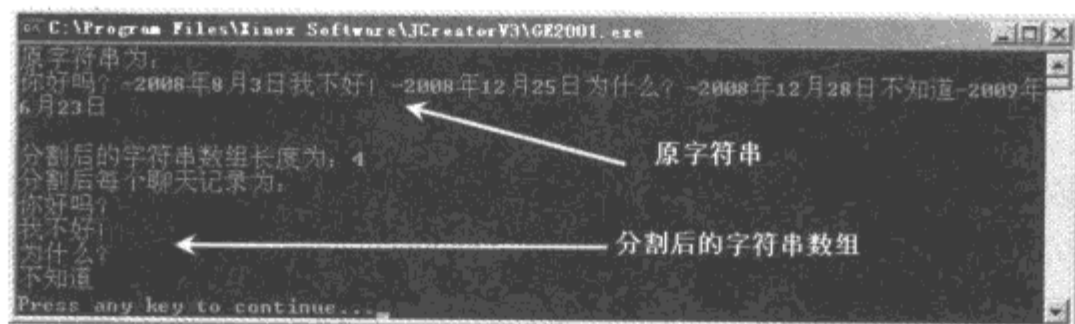


图 11-11 使用正则式进行字符串分割的运行结果图

最后需要谈一谈贪婪和非贪婪的问题，两者是进行字符串匹配查找时的不同模式。

- 贪婪模式下进行匹配时先假设整个字符串都是匹配正则式的，如果匹配不成功，则缩小范围继续验证，这样总是会按照最大限度的可能进行匹配。
- 非贪婪模式下的字符串匹配将会尽量缩小匹配成功字符串的规模，一般是从字符串起始位置开始查找，一旦匹配成功就退出此次匹配检查，类似于“见好就收”。

以下代码比较了贪婪模式和非贪婪模式对同一个字符串进行匹配的不同结果。

```

1  package wyf;
2  import java.util.regex.*;           //引入相关包
3  public class regGreedy{
4      public static void main(String args[]){
5          String patternStr = "f\\w+?f";           //非贪婪正则式
6          String patternStrGreedy = "f\\w+f";       //贪婪正则式
7          String s = "ajfnnfnnnnfda";             //目标字符串
8          System.out.println("目标字符串是: "+s);   //打印输出目标字符串
9          //非贪婪模式匹配
10         Pattern p1 = Pattern.compile(patternStr); //创建非贪婪 Pattern 对象
11         Matcher m1 = p1.matcher(s);
12         while(m1.find()){
13             System.out.println("贪婪模式下匹配到的是: "+m1.group());
14         }                                           //输出匹配结果
15         //贪婪模式匹配
16         Pattern p2 = Pattern.compile(patternStrGreedy); //创建贪婪 Pattern 对象
17         Matcher m2 = p2.matcher(s);
18         if(m2.find()){
19             System.out.println("贪婪模式下匹配到的是: \t"+m2.group()); //输出匹配结果
20         }
21     }
22 }

```

正则式中用“?”来修饰非贪婪模式的字符串匹配。以上代码的第 5 行及第 6 行就是这两种不同匹配正则式，代码运行的结果如图 11-12 所示。

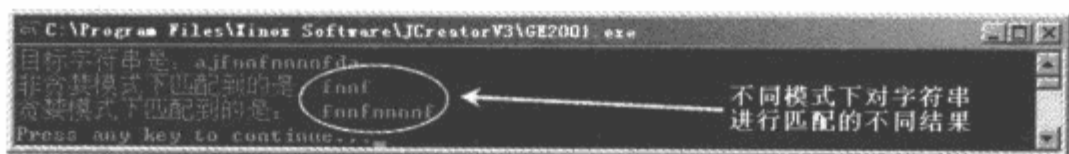


图 11-12 贪婪模式和非贪婪模式的比较

### 11.2.3 正则式在 JavaScript 中的运用

本书在前面的章节中也向读者介绍了 JavaScript 是 Java 开发人员需要掌握的一门语言。JavaScript 作为 Web 客户端最流行的动态脚本语言，当然不会忽视对正则式的支持，本小节就来简单介绍一下正则式在 JavaScript 中的运用。

“蔡佳娃，正则式在 Java 中的运用大概就是这样，你可以在以后的开发中不断积累，慢慢你就能熟练使用这个强大的字符串处理工具了。”

“是啊，我觉得这个比 SQL 的执行效率还要奇妙，而且非常锻炼脑筋呢，稍不留神开发出来的正则式就会产生难以捉摸的 bug。”

“呵呵，看来你对正则式是真的上心了，好的，趁热打铁，我们来谈谈正则式在 JavaScript 中的运用，这样你就不必局限于 Java 这一种环境了。”

“哦？JavaScript 中也支持正则式吗？”

“当然喽，而且身为 Java EE 开发人员，JavaScript 可是一门必修课呢，所以 JavaScript 中的正则式也就必须要掌握了。”

在 Web 应用中，JavaScript 多用于进行本地验证或与服务端进行动态交互，这些都需要进行一些字符串的处理。因此要想精通 JavaScript，正则式的熟练运用也就必不可少。

JavaScript 中正则式对字符串的操作也是查找、匹配和替换等。JavaScript 中有表示正则式的对象 `RegExp`。JavaScript 中的正则式也可以指明查找模式，如是否区分大小写、是否进行多行查找等。JavaScript 中的 `String` 对象也和 Java 中的 `String` 对象一样支持正则式。

正则式在 Web 开发中一个最常见的用途就是进行一些验证工作，下面举一个用正则式验证用户电子邮箱格式是否正确的例子，其代码如下所示。

```

1  <html>
2      <head>
3          <title>测试邮件地址是否匹配</title>
4          <script language="JavaScript">
5              function check()
6              {
7                  var s = mailForm.mailAddr.value;           //获取用户输入的邮箱地址
8                  var reg = /\w+(\.\w+)*@(\w+)\.\w+/;         //定义正则式
9                  if(reg.test(s))                             //判断是否匹配成功
10                     { //匹配成功
11                         alert("您输入的邮箱地址为: \n"+s+"\n 满足邮箱地址条件!! ");
12                     }
13                 else
14                     { //未匹配成功
15                         alert(s+": 不满足条件");             //显示错误提示对话框

```



```

16         }
17     }
18     </script>
19 </head>
20 <body>
21     <form name="mailForm" onsubmit="check();">
22         请输入邮箱地址: <input type="text" name="mailAddr"><br>
23         <input type="submit" name="bOk" value="验证">
24     </form>
25 </body>
26 </html>

```

- 第 8 行的“var reg = /\w+(\.\w+)\*@(\w+)\w+/;”是 JavaScript 中定义正则式的一种方法，两个“/”中间的内容为正则式，可以在第二个“/”之后附加查找模式说明。还有一种定义方法是“var reg = new RegExp(“\w+(\.\w+)\*@(\w+)\w+”,ig)”，其中“ig”是查找模式说明。
- 第 9 行的“reg.test(s)”是测试指定字符串是否匹配正则式所代表的模式的函数，返回值类型为布尔型。

上述代码的运行结果如图 11-13 所示。

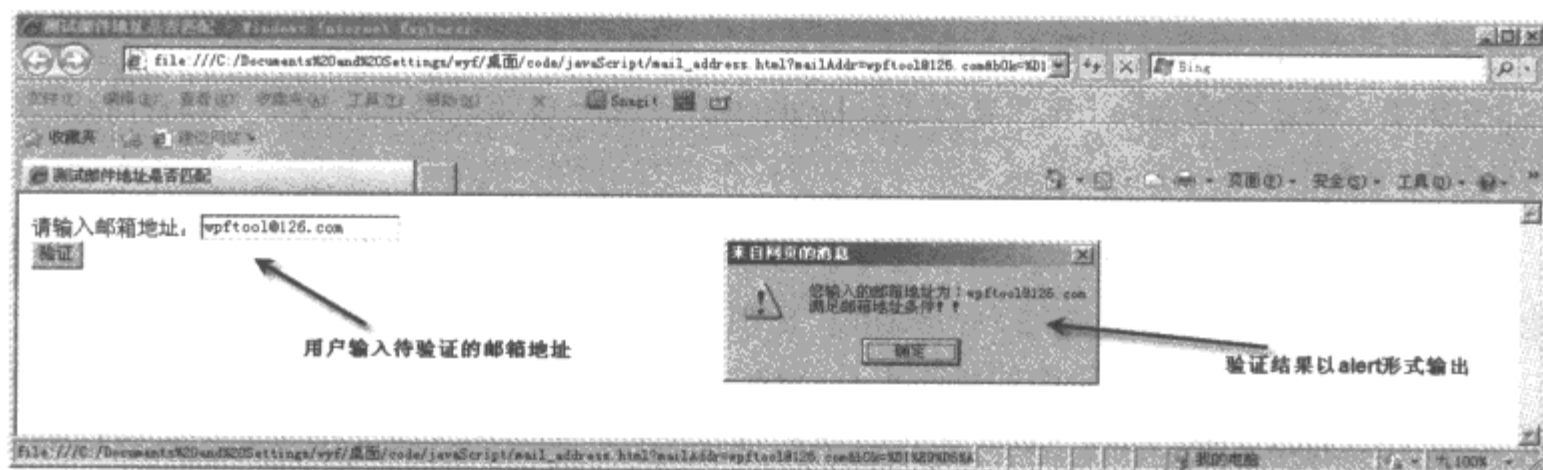


图 11-13 验证邮箱地址正确性示例

当然，本程序中验证邮箱地址的正则式只是为了拿来做一个示例，是最简单的一种做法，并没有考虑到很多东西，比如正常的电子邮件地址都是以“.xxx”结尾的，为了提高正确性，应该将正则式升级为如下形式。

```
1 \w+(\.\w+)*@(\w+)\.([com|cn|net|org])
```

这样就可以初步避免形如“wyf@wfy.com.hello”这样的邮箱地址通过验证了，但是既然最后的后缀是采用列举的方式，就有可能会有遗漏，尤其是不同国家和地域的顶级域名并不相同，如果要顺应全球化趋势，还得需要加上“|au|uk|jp|us”等。

由此可以看出，正则表达式的语法很简单，但是想要充分考虑实际情况使其实用性更好，就必须依赖开发人员的聪明才智了。对于正则式的用法，没有对错，只有完不完美。

“师兄，我觉得正则式实在是太精练了，整个二十多行的程序，核心处理的代码仅用了一行。一句小小的正则式就将情况众多的邮件地址彻底验证，真是太高妙了！”

“的确，目前为止正则式是各种字符串处理中的最佳解决方案，最明显的特征就是在代码长度上一句抵千行。不过想要把千行代码浓缩成那么精湛的一句，可是需要很高的水平哦！”

“师兄你说得太对了，我觉得正则式就是字符串处理中的贵族，能够精通正则式绝对是一件倍儿有面子的事。”

许多情况下，使用正则式都是一个非常明智的选择。运用正则式去进行字符串操作，无论是在开发时间还是执行效率及系统健壮性等方面都有着无法比拟的优势。不使用正则式的话，代码写得费事、易错，还可能有执行时的 bug，可读性也差。二者的对比如图 11-14 所示。

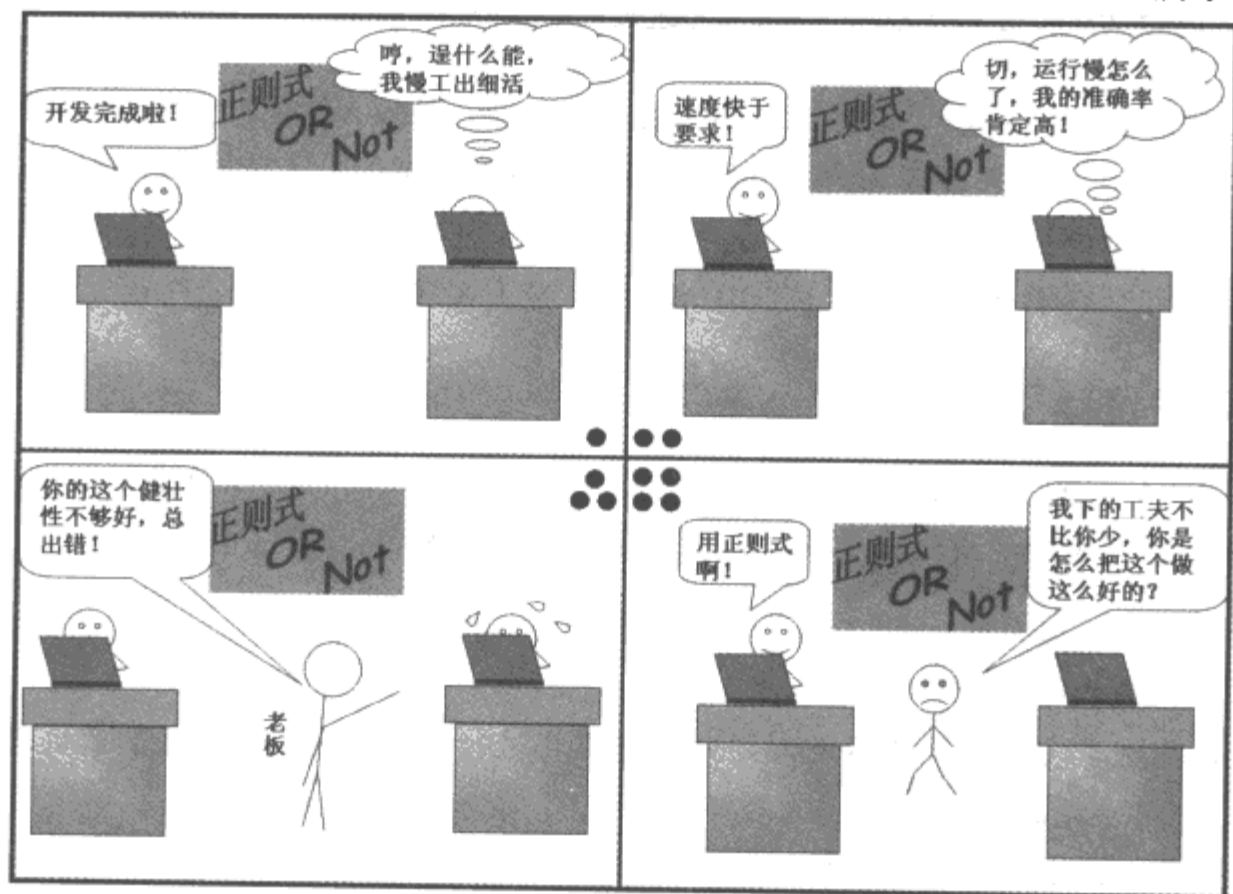


图 11-14 正则式与否的选择

JavaScript 中的正则式还有一些特性，这些特性有助于开发人员更好地发挥正则式的优势。

#### ● 捕获和非捕获性分组

不管是在 Java 中还是在 JavaScript 中，正则式进行匹配成功后是可以获得匹配成功的字符串的，这种特性使得其在字符串的替换等方面显得很有优势。但是有些时候，使用正则式只是为了验证匹配与否，并不关心匹配成功的字符串，这个时候就可以使用非捕获性分组。

在特定分组的左括号后加上“?:”就表示该分组是非捕获性分组，如下代码所示。

```
1  (?:\w+)(?:\.\w+)*@(?:\w.)*(?:\.[com|cn|net|org])
```

上述代码表明在验证电子邮箱地址是否正确时，不需要保留匹配成功的字符串。这样可以提高匹配速度和节省系统开销，对于在浏览器上解释执行的 JavaScript 来说意义很大。

#### ● 前瞻

前瞻也是为了提高效率而存在的，前瞻使得 JavaScript 在进行字符串匹配检查时先行验证某些字符，但是并不改变当前正在匹配的位置。例如需要对电子邮件地址进行某种过滤，只允许指定后缀的邮箱地址通过，可以使用前瞻技术，代码如下所示。

```
1  \w+(\.\w+)*@(\w.)*(?=\.cn)
```

这样在匹配时会首先查看邮箱地址是否满足后缀为“.cn”的条件。与前瞻对应的还有负向前

瞻，它用“?! ”表示，意义与前瞻正好相反，表示首先查看指定字符是否不满足匹配条件。

- 贪婪与非贪婪

JavaScript 中的正则式也分为贪婪和非贪婪模式，其语法与 Java 中的语法类似，也是需要在表示量词的后面加上“? ”，不加“? ”的默认情况下将会按照最大限度去匹配字符串。

## 11.3 不会用 Ant 的开发人员不是好 Developer

如果说正则式这门必杀技还是有很多初学者知道的话，那么 Ant 估计就很少有初学者了解了，但是 Ant 绝对是 Java 开发人员成为高手的路上必须征服的一座山。不比正则式和 SQL，Ant 并不能帮助开发人员写出更加高效的代码。Ant 关注的是项目源代码开发完毕之后的工作，它是一种项目构建工具。可以毫不客气地说，不会用 Ant 的开发人员不是好的 Java 开发人员。

### 11.3.1 Why Ant

构建用 Java 开发的项目时，为什么选择使用 Ant 呢？与其他的项目构建工具相比，Ant 的优势在哪里呢？本节就来将 Ant 这个强大的项目构建工具介绍给读者朋友们。

“师兄，最近我真够疲惫啊！”

“哦，熬夜玩游戏了？”

“哪啊，我发现其实上班后并不是大部分时间都在进行开发创造。就像我，最近连续几天忙着部署调试公司即将结尾的一个项目，这活可真累啊！”

“是啊，部署测试是比较累人，今天我来给你介绍一个项目构建工具，会了这个也许会让你在源代码开发结束之后的工作更加顺利一些。

“哦，那是什么啊？”

“这个工具就是 Ant。”

在使用 Java 做程序开发时，对于比较小的程序，一般的过程就是源代码开发、编译、运行等，其过程如图 11-15 所示。



图 11-15 小型程序开发流程

上述流程不算烦琐，基本上也不用什么工具辅助，开发人员可以自己点来点去就能完成。但是如果是一些大型的项目，问题就不是这么简单了。大型的项目由于分为多个模块，各个模块之间彼此依赖或通信，所以整个开发流程就复杂一些了，如图 11-16 所示。

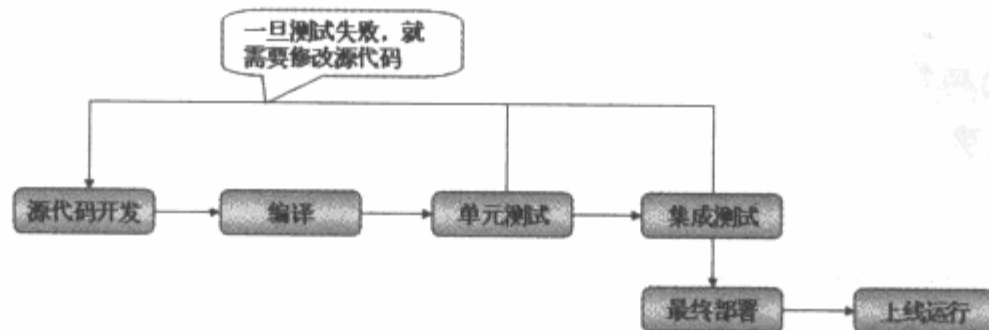


图 11-16 大型项目的开发流程

虽然只是增加了测试和部署两个阶段，但是其带来的工作量却是巨大的。做过 EJB 项目部署的人都会有所体会，每次对项目进行测试，需要进行的部署工作非常烦琐。而且不像游戏中可以保存进度，在进行测试部署的时候，一旦某个环节出了差错，只好重新再来。测试部署更像是拍电影，一旦出现“NG”，整个镜头就得从新来过。

而且这些必做的工作都是固定的，开发人员应该将精力集中到源代码的开发上来。项目构建工具就是用来完成这些没有技术含量但又必不可少的死板工作的。

“师兄，那目前市面上除了 Ant，还有什么其他的项目构建工具呢？”

“目前比较流行的项目构建工具有很多，比如应用比较广泛的 make 工具，它虽然可以构建 Java 项目，但还是主要作为 C 和 C++ 项目的构建工具。”

“哦，那还有什么啊？”

“其他的构建工具还有 GNU make、nmake、jam 等，下面我们来将这些项目构建工具和 Ant 对比一下。相比于这些项目构建工具，Ant 的优势有如下几个方面。”

- 跨平台性

Ant 是由 Java 开发的，因此就有了天生的跨平台特性，其能够在各种操作系统平台上很好地工作，不依赖于任何一种操作系统平台。而其他的构建工具一般局限于特定的平台，在跨平台这方面做得不如 Ant 好。

- 配置简单

其他的项目构建工具很多都有着复杂的配置，稍不留神就容易出错，不容易上手。而 Ant 的配置是基于 XML 的，XML 本身就是一个功能比较强大的标记语言。Ant 用 XML 的语法来约束开发人员的配置，使其更加具有灵活性和易用性，使得配置更为简单高效。

虽然纵观所有编程语言，目前 Ant 还不是使用最广泛的项目构建工具，但是当下 Java 平台中的所有大型项目几乎都是由 Ant 来构建的。而 Java 又是编程语言中的巨人，因此目前可以说 Ant 是最流行的项目构建工具。在 Java 界，Ant 已经成为一个标准的项目构建工具了。

“师兄，我就不明白了，Ant 这么厉害，那我们还要 IDE 干啥啊，它们之间有啥敌对竞争关系不？还是二者互不相干啊？”

“这不能一概而论，Java 界的强力 IDE 也就 Eclipse 和 NetBeans 两家，其中 NetBeans 本身就嵌入了 Ant 脚本。你在 NetBeans 下开发项目，总是会看到一个叫做 build.xml 的文件，这个就是 Ant 脚本，所以其实 NetBeans 就是用 Ant 来构建项目的。”

“哦，那 Eclipse 呢，我可是没看见过有 build.xml 这个文件啊！”

“Eclipse 本身的项目构建虽然不是基于 Ant，但是 Eclipse 完全支持使用 Ant 脚本替代其自身的构建工具去构建一个项目。”

“哦，是吗？我回去得试试。”

“其实，Ant 还有一个最大的作用是可以作为这两种 IDE 的桥梁，比如你在 NetBeans 下开发了一个项目，却由于种种原因需要迁移到 Eclipse 下或者需要将项目从 Eclipse 下迁移到 NetBeans 下。这时候通过 Ant 进行迁移将会是开销最低、最快捷的一种迁移方式。”

作为一种辅助开发的工具，IDE 在项目开发和编译尤其是调试等环节做得非常出色，为开发

人员带来了很大便利。同时 IDE 也提供了一些项目构建方式用于测试和部署，但是很多时候提供的功能比较有限，并不能满足所有的需求。而且在 IDE 上开发的项目移植性并不好，往往在一种 IDE 上开发的项目很难拿到另一种 IDE 上调试开发。

而 Ant 脚本得天独厚的跨平台特性及广泛认可就是 IDE 工具梦寐以求的合作对象了。通常在一些项目的构建工作中，可以自己用 Ant 开发，也可以让 IDE 代劳。但是遇到 IDE 无法胜任的工作，就可以由 Ant 来进行项目的构建。例如遇到 NetBeans 没有的功能时可以手动修改 build.xml 来实现一些特殊的需要，而不必等着 IDE 升级换代了。

而且 Ant 可以方便地和 JUnit 框架结合，后者用于单元测试。所以对于 Ant 这个 Java 项目构建的标准，开发人员不掌握是万万不行的。

### 11.3.2 Ant 初体验

前面的小节介绍了 Ant 的特性，读者朋友大概也已经认识到 Ant 对于 Java 开发人员的重要性了。本小节将介绍 Ant 工具的基本用法，算是跟这个强大的项目构建工具混个脸熟。

“蔡佳娃，耳听为虚，眼见为实，接下来我们就给出一个使用 Ant 构建应用程序的小案例，让你也感受一下 Ant 的基本用法。”


“好啊，让我来体验一把小蚂蚁工具厉害，呵呵。”

“你肯定会感受到它的便捷的。”

本案例使用 Ant 构建一个简单的 Java 应用程序，其步骤如下。

#### 1. 下载 Ant 安装包

从网上下载 Ant 的安装包 apache-ant-1.7.1-bin.zip，并将其解压缩到磁盘的某个位置，如“C:\apache-ant-1.7.1”。

 **提示** Ant 属于 Apache 软件基金会，免费使用，无须安装，解压缩后即可运行，下载地址为 <http://ant.apache.org/>。

#### 2. 设置环境变量

解压缩完成后需要设置两个环境变量，第一个是添加 JAVA\_HOME 环境变量，该环境变量指向 JDK 的安装目录；第二个是修改 PATH 环境变量，在其中加上 Ant 安装目录下的 bin 目录，如“C:\apache-ant-1.7.1\bin”。配置好环境变量之后打开命令行窗口（console），输入 ant 命令后如果出现如图 11-17 所示的信息则说明安装配置成功。

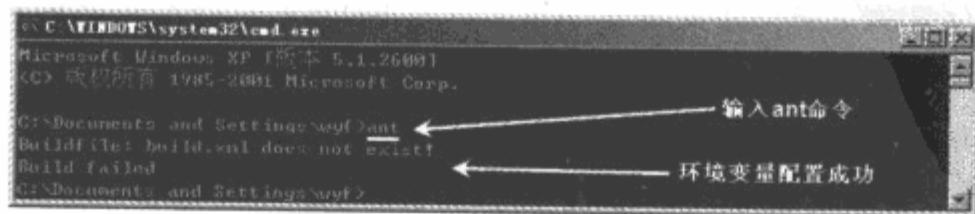


图 11-17 ant 配置环境变量成功

#### 3. 开发具体项目的源代码

开发需要被 Ant 构建的应用程序的源代码，本案例中将以一个最简单的名为 HelloAnt 的 Java



类作为被构建的应用程序，源代码极其简单，不予列出。为方便 Ant 脚本进行构建，在项目文件夹的根目录下面设立“src”和“classes”两个文件夹，分别用来存放源代码和编译之后的类文件。

#### 4. 编写项目构建文件 build.xml

在项目文件夹根目录下建立一个名为 build.xml 的文件，并在其中输入如下 XML 代码。

```

1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <project name="WYF_HELLO" default="RunHello"><!-- 声明工程名称，指定默认目标为 RunHello-->
3      <target name="CompileHello">                <!-- 定义一个名为 CompileHello 的 target -->
4          <javac                                    <!-- 编译源代码的任务 -->
5              destdir="classes"
6              srcdir="src"
7          >
8      </javac>
9  </target>
10 <target name="RunHello" depends="CompileHello"> <!-- 定义一个名为 RunHello 的
target-->
11     <java                                    <!-- 执行应用程序的任务 -->
12         classname="wyf.HelloAnt"
13         classpath="classes"
14     >
15     </java>
16 </target>
17 </project>

```

- 每个工程的构建文件都是由一些目标（target）组成的，如代码第 3 到第 9 行、第 10 到第 16 行就使用<target>标记分别定义了两个目标。
- 在目标之中定义的是具体的任务，如名为“CompileHello”的目标中要执行的任务就是编译(javac)。其中“destdir”表示编译后类文件的存储位置，“src”表示源代码的存储位置，这些位置路径都是可以自己定义的。
- 在工程的构建文件中，目标和目标之间是可以指定依赖关系的，如项目执行必须在编译之后，则“RunHello”目标的执行必须依赖于“CompileHello”目标。代码中第 10 行的“depends”就表明了这种依赖关系。

#### 5. 运行

编辑好项目构建文件之后，打开命令行窗口，将路径设置为项目所在路径，输入 ant 命令，Ant 将会根据构建文件的配置对项目进行编译和运行，其效果如图 11-18 所示。

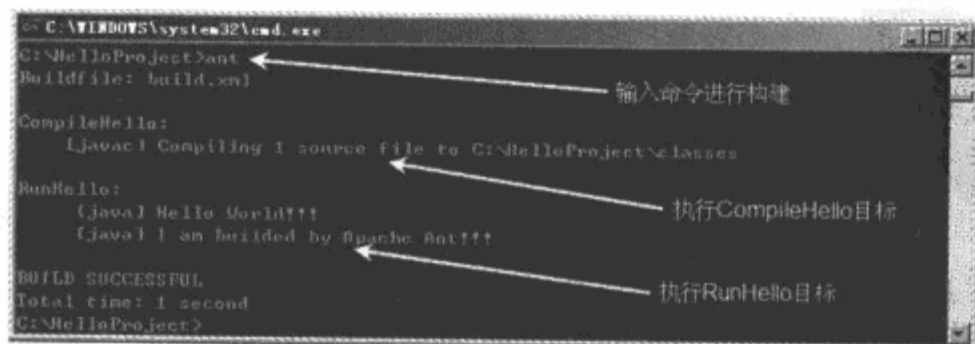



图 11-18 ant 构建项目图示

其中执行 RunHello 目标输出的两句话是在源代码中定义的。这只是 Ant 构建应用程序的一个小例子，只定义了编译和运行两个比较简单的目标，其他诸如打包、部署等目标并未涉及。本例只是让读者对其有个初步的体验，有兴趣的读者可以进一步学习。

 **提示** Ant 的知识远比本节所举的例子复杂，在此向有兴趣研究 Ant 脚本的读者朋友推荐一本讲解 Ant 比较全面的好书：Erik Hatcher 和 Steve Loughran 编写的《使用 Ant 进行 Java 开发》。

## 11.4 浅谈设计模式

设计模式算是软件开发中和算法比肩的高深层次了，任何一个高级的 Java 开发人员都必须对设计模式有很深的了解，否则根本无法应对摆在面前的高端问题。而想要成为武林高人的江湖小生们也必须沉静下来，潜心研究设计模式以提高内力的修为。

就像本节标题所取的那样，对于承载了顶尖开发思想的设计模式，本书只能浅谈。本节将谈谈设计模式对于开发人员的重要性，然后将会选择几种常用的设计模式做一个简单的介绍，以期读者朋友们能够对设计模式有一个全面而又不失具体的了解。

### 11.4.1 设计模式的重要性

“哎，师兄，你还有什么必杀技啊，尽情传授给师弟我吧！”

“呵呵，师兄我又不是一个源源不断的知识库，我给你讲了这么多必杀技，我也只剩下一个杀手锏了，那就是设计模式啦！”

“设计模式？很高妙的东西啊，MVC 应该算是一个吧？”

“对，MVC 只是比较广为人知的一种设计模式，不过你可以从 MVC 模式中体会到设计模式带来的好处。”

“嗯，有那么一点，不过师兄你还是跟我系统地讲一下吧，哈哈。”

设计模式，就是前人在大量开发实践中总结出来的解决某类问题的一种高效而精练的方法，设计模式是用来指导开发的。正如中华民族几千年下来沉淀出了博大精深的文化，设计模式就是软件开发这几十年间沉淀出的精华思想。

设计模式在开发中起着很重要的作用，如果不晓得设计模式的存在而闭门造车，只会让自己的产品在设计模式的精华思想面前相形见绌。而且很多的设计模式设计精巧，并不是一般人能够轻松想出替代方案的。因此对于设计模式要深入理解和领悟，以免在实际开发中出现问题。

目前市面上流行的设计模式大概有 23 种，除了常见的 MVC 设计模式，还有诸如抽象工厂模式、适配器模式、观察者模式等。

“师兄，那你说说设计模式和算法有什么区别呢？”

“算法也是一种非常高深的思想，不过算法研究的更多是一些具体固定的方面，其应用的范围也比较狭窄，如一个 JPEG 图像压缩算法只能用于进行图片压缩。”

“是啊，算法在解决一些微观问题的时候比较厉害啊。”

“的确，而设计模式所要解决的目标问题一般就是尺度大一些的问题了，它一般不会关注具体的实现细节。设计模式将主要将精力放在如何设计整个项目的结构以满足需要，一般不依赖于某种具体的开发平台。”

“也就是说设计模式和算法的应用领域不同喽。”

“可以这么说，设计模式主要用来解决宏观问题，而算法则更关注于如何使微观的具体问题解决起来更加高效。”

打个比方来说，算法应该是如赵子龙般骁勇善战的猛将，而设计模式则是如诸葛亮般运筹帷幄的大军师。猛将负责披坚执锐，浴血奋战拿下胜利，这就像是算法在解决某个具体问题；而军师则负责统筹大局，决定这场仗该如何打，布什么阵、出什么兵，这就像是设计模式的角色。算法和设计模式一个研究兵器，一个研究兵法，当然二者一个也不能少，否则难打胜仗。

设计模式最大的特点就是可以复用，就像兵法一样，在这里打了胜仗，在外界条件相似的情况下仍然可以继续创造胜利。设计模式应该是本章列举出必杀技中需要内力修为最高的一个了，在后面的几个小节中，本书将会着重介绍一些常用的设计模式，以加深读者对设计模式的感性认识。

#### 11.4.2 MVC 设计模式

MVC 算是设计模式家族中知名度最高的一个了，目前几乎所有的 Web 应用开发都是基于 MVC 设计模式的。因此要介绍设计模式，肯定不能错过 MVC。

MVC 设计模式的工作原理如图 11-19 所示。在 MVC 设计模式中，控制器（Controller）负责接收（监听）所有请求（消息），模型（Model）用于表示要被显示的数据和操作，视图（View）负责将模型以某种形式呈现给用户。

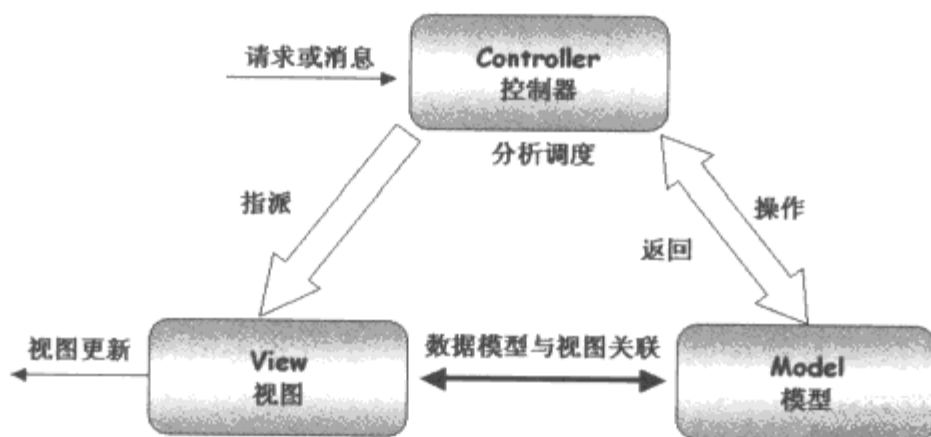


图 11-19 MVC 设计模式的工作原理

当请求或消息到来时，控制器对其进行分析，并决定调用哪个业务逻辑去修改数据模型，修改完成后将会指派用何种视图方式反馈给用户；某个视图得知自己将要被用来显示数据，就会与数据模型取得联系，将数据呈现出来。

简单来说，MVC 设计模式有如下优点。

- MVC 设计模式将控制器、模型和视图分割开来，使得各个模块的耦合度降低。这样可以各司其职，处理好自己负责的部分。如负责视图的开发人员只需要研究如何提供更优质的数据表现形式即可，无须操心复杂的业务流程。

- MVC 设计模式具有类似“模块化，可插拔”的特性，由于控制器、模型、视图之间相互独立，可以在需要时将其中一个部件进行替换而不需要对其他部分做太多的改动。这就避免了“牵一发而动全身”，使得系统的可维护性更好。

MVC 设计模式的一个主要应用就是进行 Web 开发，主流的 Web 开发框架如 Struts、JSF 等都是基于 MVC 设计模式的，在一个简单 Web 应用中的 MVC 设计模式如图 11-20 所示。

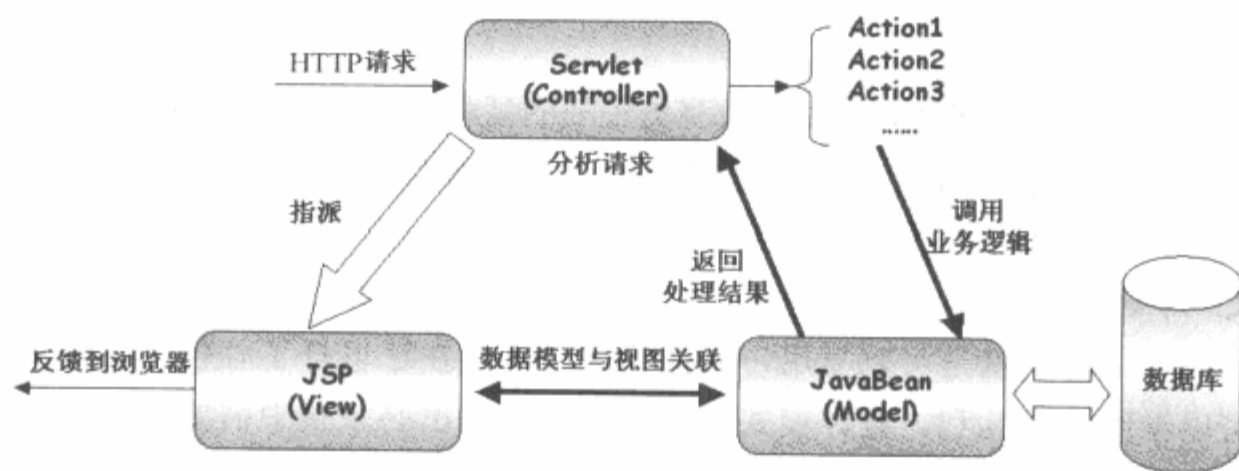


图 11-20 Web 开发中的 MVC 模式

当 HTTP 请求到来后，Servlet 将对其进行分析，然后决定需要执行哪一种动作(Action)，随即调用 JavaBean 对数据进行操作，其间 JavaBean 可能需要访问到数据库。JavaBean 处理完业务逻辑后会返回到 Servlet，Servlet 根据其返回的结果决定应该指派哪个视图去显示结果，被指派的视图再与数据模型进行关联，将结果反馈到浏览器。

“师兄，MVC 果然是 Web 开发中的不二选择啊！”

“呵呵，不过 MVC 也有它的缺点，其中一个就是可能会降低系统的性能。因为将系统分层必然会使得原本直接可达的操作需要走一些固定的步骤，比方说数据库的连接，没有 MVC 的话 Servlet、JSP 都可以直接访问，而有了 MVC 则需要调用相应的 JavaBean 才行。”

“对啊，不过相比于 MVC 设计模式带来的好处，这点性能损失也是值得的吧？”

“嗯，的确是这样，不过 MVC 设计模式还有一个算不上缺点的缺点，那就是 MVC 虽然看起来很容易，但很多开发人员对它的实现却很难符合实际 MVC 的结构标准。”

“哦，这话怎么讲？”

“比如说开发人员很容易将控制器、模型和视图三者的分工混淆，往往会造成玩忽职守、鸠占鹊巢的混乱现象，那样的‘MVC’可是很糟糕的。”

在实际工作中，很多开发人员都知道要用 MVC，但是却只是在代码中“体现”了这个结构，并不是真正地实现了 MVC 设计模式。很多情况下只是实现了披着 MVC 设计模式外衣的一些反模式，比较极端的就是“神奇 Servlet”反模式和“神奇 JSP”反模式。在这里“神奇”可不是高妙的意思，而是玄乎其神、没有价值的意义。

“神奇 Servlet”反模式的结构如图 11-21 所示，其中基本看不到 JSP 的身影。Servlet 在担任控制器角色负责调度的同时，也不辞辛劳地担负起了数据呈现的任务。这种情况产生的原因或许是因为 JSP 比 Servlet 技术出现得晚，而 Servlet 在 JSP 出现之前也是这么负责视图的，但随着时间的推移还照老路走就大大不妙了。

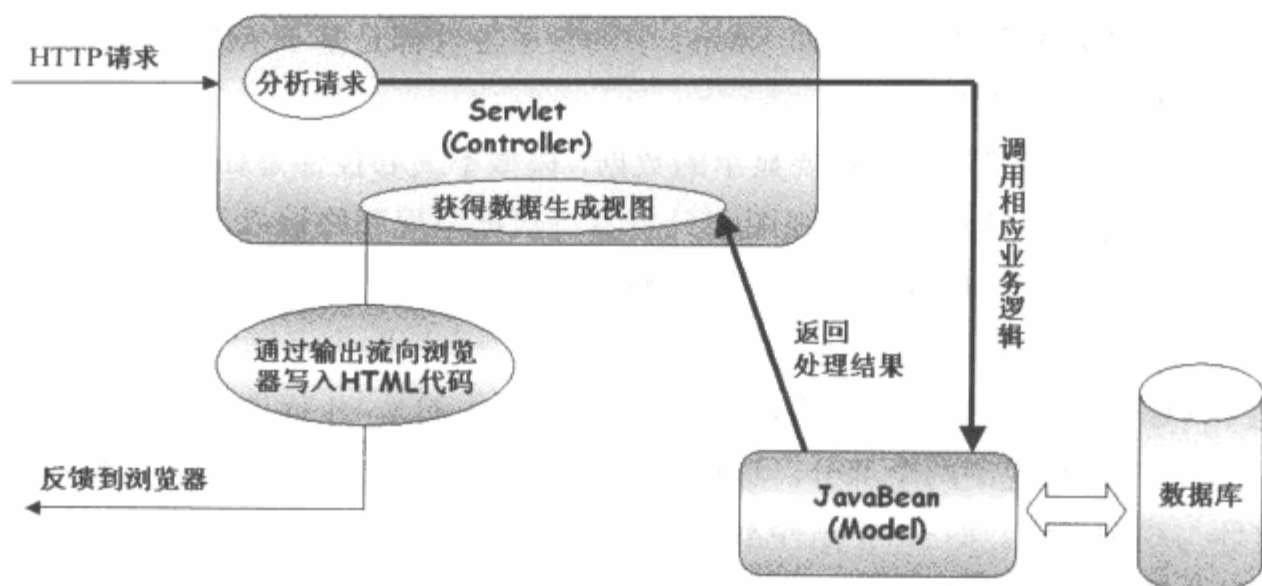


图 11-21 “神奇 Servlet”示意图

“神奇 Servlet”反模式中控制调度和视图显示全部集中到 Servlet 这一端，使二者分离度降低，不利于业务功能的扩展。而且使用 Servlet 做视图开发相当麻烦，效率低下。最重要的是会使得 MVC 的核心部分——控制器臃肿不堪，这必然会大大降低系统的可维护性。

相比之下，“神奇 JSP”则是对 JSP 这个比较年轻的技术过分使用了。“神奇 JSP”反模式的结构如图 11-22 所示。在这种模式下，所有的页面请求都只会送到对应的 JSP，JSP 揽下控制器的工作，既负责通知 JavaBean 处理具体业务逻辑，又负责视图显示。

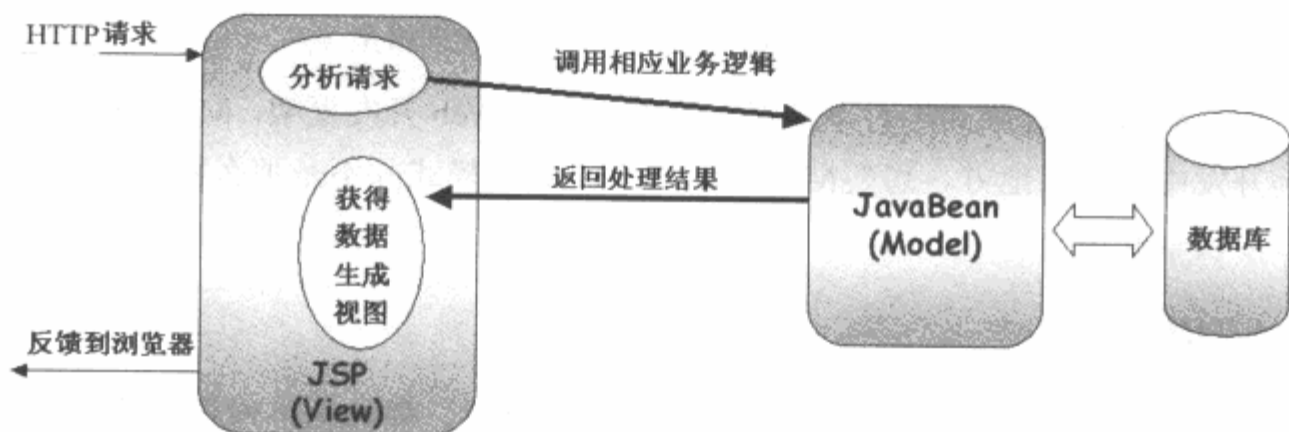


图 11-22 “神奇 JSP”示意图

“神奇 JSP”反模式仍然没有将 Web 应用中的控制调度和视图显示工作合理分开，只不过这次是将工作全部集中到 JSP 这端而已。这种在 HTML 标记中嵌入大量 Java 代码的做法显然难以为继，而且“神奇 JSP”模式不利于对业务流程的集中控制，使得扩充新业务功能十分困难。

**提示** “神奇 Servlet”与“神奇 JSP”反模式只是极端的情况，实际中很多开发人员只是在一些局部地方由于设计不慎变成了两种反模式之一，这点读者朋友们也要多注意。

“哎，看来想要掌握好 MVC 设计模式进行 Web 开发，还真不能只浮于表面啊！”

“那是当然，不过你可不要认为 MVC 模式仅仅是个用于 Web 应用的设计模式哦。MVC 可是个‘放之四海而皆准’的设计理念啊。”

“怎么？MVC 除了用在 Web 开发中，还能用在别的地方吗？”

很多开发人员都错误地认为 MVC 设计模式就只是“Servlet+JavaBean+JSP”，其实 MVC 设计



模式的思想在很多地方都有体现。例如 Swing 中的很多控件都是基于 MVC 设计模式实现的，这样可以大大提高开发的灵活性。

在 Swing 控件中，模型就是控件要显示的数据，模型不关心控件是如何将自己呈现给用户的，而主要负责对数据进行存储和管理。视图则只负责将数据呈现到终端以及将用户事件传递给控制器，视图可以有多种形式，但视图永远不能影响到模型的内容。控制器依然主管统筹调度，一旦用户进行了相关操作，控制器就会根据不同的动作对模型和视图进行不同的修改。

与其他 MVC 设计模式有所不同的是，Swing 控件采用的是 UI 代理的模型，如图 11-23 所示。UI 代理集成了控制器和视图两个部分，这样一来 UI 代理既可以呈现视图，又可以处理事件，并作为一个整体与模型通信。如 JList 控件不仅可以显示数据，还接收并分析用户的鼠标单击等操作以调用不同的事件处理方法进行必要的数据库变动或视图转变。

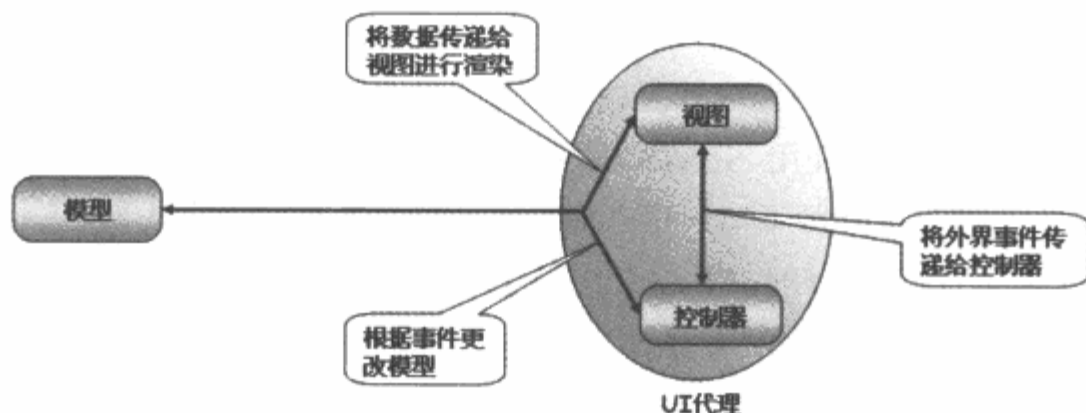


图 11-23 Swing 控件中的 UI 代理模型

由此可见，MVC 设计模式绝对不是那么简单的一个 Web 开发框架，MVC 所包含的设计思想可以运用到开发中的很多地方。想要彻底掌握 MVC 设计模式并不是那么容易的，开发人员需要在实践中多多摸索和体会。

### 11.4.3 单例模式

“蔡佳娃，下面我们来谈一个比较容易理解的设计模式，那就是单例模式。”

“单例模式？单一实例吗？”

“呵呵，差不多吧。单例设计模式就是某一个类在运行的时候只能够有一个对象，再次创建对象时返回的也是已有的那个对象。”

“这么小气啊！那单例模式有什么用呢？”

单例模式（Singleton）也是应用得比较多的一种设计模式，在软件开发中有时需要对某个类的对象数量加以限制。比如在 GUI 程序设计中，某个经常用到的窗口对象只需要一个就够了，没必要每次用到就去创建新的窗口。而如果只创建一个对象将引用随便传来传去又不方便，这时候采用单例模式就是个比较好的解决方案了。

使用单例模式可以很好地节约系统资源，尤其是当对象比较大时，对系统开销的节约效果将会更加显著。同时由于系统的多个地方共享一个对象，在一定程度上便于控制。下面给出一个 Java 下实现单例模式的例子，其代码如下所示。

```
1 package wyf;
2 public class Singleton_Sample{                                //定义一个单例模式的类
```

```

3      public static Singleton_Sample mySingleton;           //定义唯一的对象
4      public static Singleton_Sample getInstance(){           //定义对象工厂方法
5          if(mySingleton == null){
6              mySingleton = new Singleton_Sample();           //若不存在对象，则创建一个
7          }
8          return mySingleton;
9      }
10     private Singleton_Sample(){                             //定义构造器，是私有的
11         System.out.println("创建了一个 Singleton 对象！"); //打印提示信息
12     }
13     public static void main(String args[]){                 //主方法
14         Singleton_Sample s1 = Singleton_Sample.getInstance();//创建第一个对象
15         Singleton_Sample s2 = Singleton_Sample.getInstance();//创建第二个对象
16         //判断两次创建的对象是否为同一个对象
17         System.out.println("这两次创建的对象是" + (s1==s2?"相同的！":"不相同的！"));
18     }
19 }

```

- 第 3 行声明了一个 Singleton\_Sample 类的静态成员，这就是指向那个唯一对象的引用。
- 第 4 行声明了一个静态方法，这个方法返回一个单例模式类对象的引用。其首先判断对象是否已经创建，如果已存在则直接返回其引用，如果不存在则创建一个新对象并返回其引用。
- 第 10 行为 Singleton\_Sample 类的构造器，该构造器是私有的，表明外面无法直接调用构造器来创建对象。

编译运行上述代码，其结果如图 11-24 所示。



图 11-24 单例模式示例的运行结果

“单例模式果然很好用啊，节约才是美德。”

“不过使用单例模式时也要注意啊，否则会节约不成反浪费哦。”

“哦？还会产生浪费？”

“单例模式下的对象一般生命周期比较长，而有时候一些其他对象会注册到这个单例模式对象中。比如通过添加到单例模式对象的某个 HashMap 成员中，这时候如果不注意在使用完成后处理掉注册对象的话，就会成为我们之前讨论过的‘小肥猪’内存漏洞了。”

#### 11.4.4 最终守护者模式

“师兄，下面我们谈什么设计模式啊？”

“还谈啊！你小子不饿是吧，师兄我还要吃饭呢。”

“师兄，还早呢。再谈一个我们就去吃饭，我请客。”

“好吧，我们就再谈一个，剩下的你可以去看书体会，最后谈的这个叫做最终守护者模式。”

“最终守护者，守护谁啊？”

“听我讲完你就明白了。”

Java 的内存管理大家都很清楚是采用垃圾收集器的方式，而对象被清除出内存之前，可能还有一些“遗愿未了”，需要进行一些扫尾的工作。在 C++ 中这种工作是通过析构函数来实现的，而在 Java 中这些代码可以放在对象的 `finalize` 方法中去完成。`finalize` 方法是由每个对象继承自 `Object` 类的，如果对象希望在被收集前执行特定的代码，就需要重写继承的 `finalize` 方法。

但是重写 `finalize` 方法的时候除了要编写自己的清理代码之外，还需要调用父类的 `finalize` 方法，因为子类的对象同时也是一个父类的对象。创建子类对象时级联调用了父类的构造器，那么清理子类对象时，父类对象的清理代码也应该执行。

但是不像构造器的级联调用是自动的，`finalize` 方法的级联调用需要开发人员主动去做，这项工作很容易被忽视并因此带来严重的问题。而使用最终守护者模式，就可以解决这类问题，下面给出一个使用最终守护者模式的例子。

```

1  package wyf;
2  //定义实现最终守护者模式的类
3  class FinalGuarder_Father{
4      private Object guarder = new Object(){           //定义守护者对象
5          public void finalize() throws Throwable{ //重写守护者对象的 finalize 方法
6              System.out.println("FinalGuarder_Father 类对象被垃圾收集器收集， "+
7                  "将执行相应的清理工作！");           //打印提示信息
8          }
9      };
10 }
11 //定义继承自最终守护者模式类的子类
12 class FinalGuarder_Son extends FinalGuarder_Father{
13     public void finalize() throws Throwable{           //重写 finalize 方法
14         System.out.println("FinalGuarder_Son 类对象被垃圾收集器收集， "+
15             "将执行相应的清理工作！");           //打印提示信息
16     }
17 }
18 //定义主类
19 public class FinalGuarderSample{
20     public static void main(String args[]){           //主方法
21         FinalGuarder_Son fgs = new FinalGuarder_Son();
22                                                     //创建最终守护者模式类的子类对象
23         fgs = null;                               //将对象置为垃圾
24         System.gc();                               //申请垃圾收集
25         try{                                       //主线程休眠
26             Thread.sleep(100);
27         }
28         catch(Exception e){                       //捕获可能抛出的异常
29             e.printStackTrace();                   //打印异常信息
30         }
31     }

```

- FinalGuarder\_Son 类继承自 FinalGuarder\_Father 类，并且重写了其 finalize 方法，但在重写时没有进行 finalize 方法的级联调用。
- FinalGuarder\_Father 实现了最终守护者模式，其私有变量 guarder 就是守护者。当子类对象被垃圾收集器收集时，父类的成员 guarder 也将成为垃圾被收集，在被收集的时候就会执行守护者 guarder 的 finalize 方法，这样就实现了自动级联调用的意图。
- 需要注意的是守护者成员 guarder 必须是 private 的，这样才能保证当其守护的对象成为垃圾时，其自身也成为垃圾而不受外界干扰。

上述代码的执行结果如图 11-25 所示。

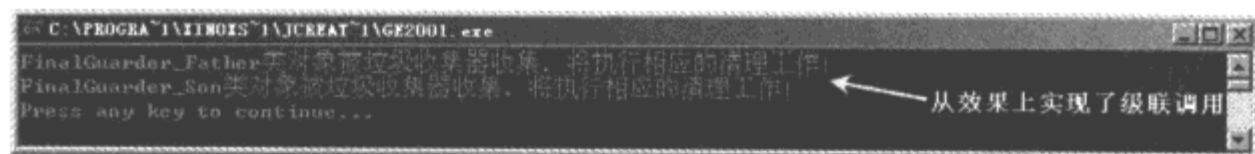


图 11-25 最终守护者模式运行结果

**提示** 设计模式除此之外还有很多种，本书在此向有兴趣深入研究设计模式的读者推荐一本经典书：《设计模式——可复用面向对象软件的基础》。

## 11.5 本章小结

本章在已经熟练掌握 Java 的日常开发的基础上，介绍了一些 Java 开发技术中的必杀技，由于篇幅所限，本书对于每一种必杀技并没有做很细致的讲解，只是着重探讨了其重要性和基本原理，这些知识如果掌握了的话，对于 Java 开发人员来说就是如虎添翼。

# 第 12 章 新锐兵器谱

任何一种行业如果失去了发展和创新，也就失去了生命力。IT 界亦是如此，“江山代有才人出，各领风骚数百年”，Java 的车轮轰隆隆永不休止地向前转动。现如今的主流技术在当年也许曾被称为“乳臭未干”，而很多开始崭露头角的新技术也很可能就是未来 Java 车轮转动的发动机。

“长江后浪推前浪，前浪死在沙滩上”，这句改装诗说明了一个残酷的道理，在技术飞速发展的 IT 界，如果不学习新技术，很有可能就会被拍到沙滩上，驱逐出 IT 的海洋。本章就来介绍当今 Java 江湖中最新锐、最酷、最炫的兵器，让读者朋友们先睹为快。

## 12.1 面向服务的体系架构（SOA）

其实，面向服务的体系架构并不能算是一种新锐兵器，SOA（Service-Oriented Architecture）理念早在很多年前就已经被提起，但是直到最近几年 SOA 这点星星之火才得以燎原。本节就来向读者介绍这个在软件开发界呼声越来越高的软件体系架构。

### 12.1.1 对面的 SOA 看过来

“师兄，这个周末去打篮球吧！看看你我荒废已久的球技是否还能驰骋球场。”

“我不去了，最近买了一本讲新技术的书，准备这个周末研究一下。”

“哦？师兄你这么牛了还在学习呀，搞得我都无地自容了。算了，我也不去了，和师兄一起好好学习好了。师兄你给我指几条明路呗。”

“呵呵，难得你这么爱学习，我就把目前市面上比较新的技术给你做个介绍吧。”

“太好了，那我们首先讲什么呢？”

“我想想看……好吧，就讲 SOA 吧，这个比较悬乎，看能不能把你给忽悠了。”

介绍何为 SOA 之前，有必要提一下目前软件产品在 IT 行业中的性质。从有 IT 历史开始，软件技术的发展经历了很多阶段。发展到现在，可以说软件已经成为一种 IT 资产而存在了。

对于任何机构团体或个人，如果想要获得一款软件并使用它，一般只有两种途径：第一种就是花钱买过来；第二种就是自己组织人手开发或者雇人开发。除了购买、开发费用，两种方法都有可能需要额外的硬件开销，因为有些软件需要有其独特的硬件平台或系统平台，买得起马得配得起鞍，于是有些时候软件成为了一笔巨大的资产投入。

“可是师兄，我觉得这样也很合情合理啊，开发出来的软件如果不拿出来卖，那开发人员每天吃代码为生吗？”

“把软件当做资产来对待，对于一些大公司还行，因为大公司有实力，可以自己做系统，自己买服务器。而对于小公司或个人来说就很不划算了。打个比方来说，你由于公司业务的需要，要进行一些工程预算，而市面上的工程预算软件很贵，而且这种软件就算花大价钱买来，或许除了



这次也不会再用第二次，显然是比较浪费的。”

“听你这么一说，还真是啊。”

“这种状况有点像以前人们对于 DVD 的感觉，人们希望看的是 DVD 里面的电影，而为了达到这个目标，不但要有 DVD 盘还需要昂贵的 DVD 播放机，所以就有一些人退缩了。”

“那怎么办呢，要是不看不行呢？”

“那就租着看呗，不管是 DVD 播放机还是 DVD 本身，都可以租嘛。这不就很好地解决问题了！这种办法在一定程度上和 SOA 思想不谋而合。租着用，是对 SOA 的一种最通俗的说法了。你可以从这里入手更全面地了解 SOA 的思想。”

其实软件产品就应该作为一种服务而不是产品来流通，很多时候企业或个人大费周章地进行购买硬件、架设服务器等工程，只是为了享受到最终的软件服务。而若想走到最后一步，前面的铺垫又不能没有。其实，人们用电的方式或许可以为解决这个问题指一条明路。

国家电网覆盖面很广，供电公司对所有家庭都提供了电线接入，人们想用电的时候就直接用，只要按照度数缴纳费用就行。如果用电量很大，如国防等特殊机构，可以自己造发电厂。而对于普通用户根本不需要自己购买发电机，不需要建造基础设施就可以用上电，如图 12-1 所示。

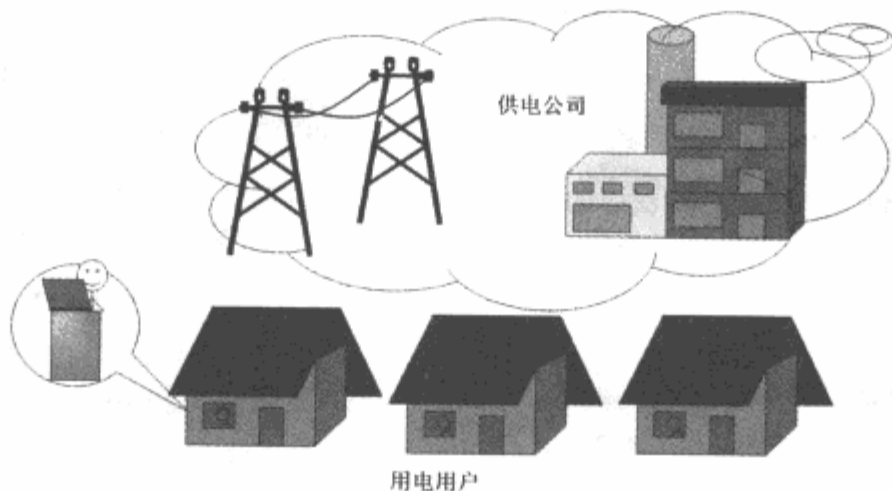


图 12-1 享受电力服务图

软件开发产业能不能也采用这样的思路呢？作为软件运营商，把软件做出来，放到一个平台上（比如 Internet），并提供一些通用的接口，让需要的人去使用，并支付一定的费用，就像用电一样方便。这就是 SOA 的基本思想。

客户对软件的需求说白了就是一种服务，SOA 提供的就是这种服务，而不是商品。用户可以租用一些服务来完成自己的目标，如进行复杂的工程计算等。这就免去了基础设施的建设，因为软件服务提供商已经将必要的如硬件服务器等基础设施建设好了。

“师兄，听你这么一说，SOA 的思想还真不赖啊！”

“是啊，SOA 思想把软件作为一种服务存在，所有软件功能不管大小都是服务，对于终端用户来说，可以直接租用，坐享其成。这样免去了从头做起，成本也就大大降低，而软件服务供应商也可以通过廉价的服务而不是高昂的软件产品或服务器赢得更广泛的用户群。”

以上探讨的是 SOA 思想给软件开发者和用户带来的好处，其实 SOA 也可以运用在软件开发商之间，这里以美国一家财务软件公司的成长过程为例来说明这个问题。在开发财务软件的过程中不可避免地需要进行税务计算，美国的税率不仅各个州不一样，同一个地方的税率变化也很快，

而且会不断产生新的合法避税的条款，废止一些旧的条款，因此掌握实时的税率计算方法对于每个财务软件公司都很重要。

而就在其他财务软件公司奔波于软件功能开发和维护时时变动的税率之间而忙得焦头烂额时，一家特殊的财务软件公司成立了。该公司只做税务方面的研究，不卖成品软件，而只是与其他大量财务软件公司合作，将自己公司专注的税务软件作为服务租给其他公司，这种新式的理念很受欢迎，这家公司也就慢慢壮大起来了。

可见企业与企业之间也可以通过提供服务来进行业务上的往来，有了 SOA 之后，可以跨企业、跨机构、跨客户地把通用的东西提取出来包装成服务。不过这还不是 SOA 的极限，在一个企业内部的大型系统之间，SOA 也有着舍我其谁的必要性。例如，旧的企业信息化模型如图 12-2 所示。

这种企业的信息化模型将每种业务功能隐藏在不同的应用程序中，形成了软件资产的“竖井”，这些“竖井”只提供私有的接口来服务于用户。“竖井”的信息模型给系统和系统之间的互联互通带来了不便，使得企业内部的信息管理分而治之，不利于整体管理。

而如果企业所部署的信息化软件是面向服务的，那么除了系统自身运作之外，还可以通过封装的方式暴露出一些服务点。通过这些基于 SOA 架构的服务点就可以进行非常高效的互联互通了。同时这些暴露出的服务点又可以通过一些流程的控制和编排，形成新的业务流程，如图 12-3 所示。

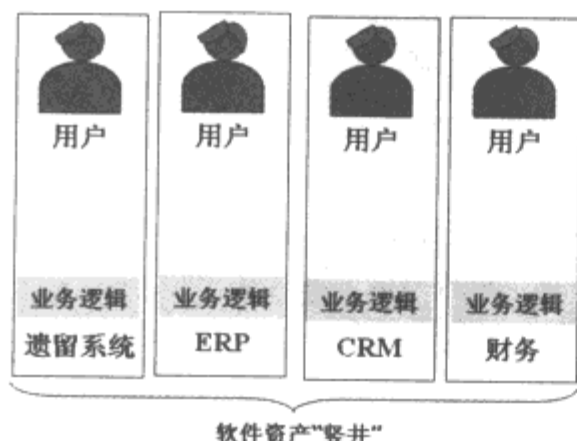


图 12-2 旧的企业信息化模型

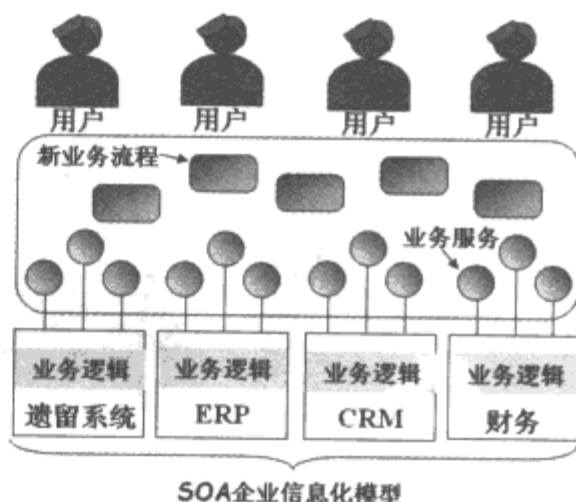


图 12-3 SOA 架构的企业信息化平台

在 SOA 的世界里，由于软件都是以服务的形式存在的，可以很方便地对其进行编排，通过对一系列服务点进行输入输出的关联和流程控制，可以形成新的服务点。而这些新的服务点又可以继续和其他服务点进行组合编排，如图 12-4 所示。一个 SOA 的平台就像是汽车的装配车间，通过将各种零件进行组装，生产出符合各种需求的车型。

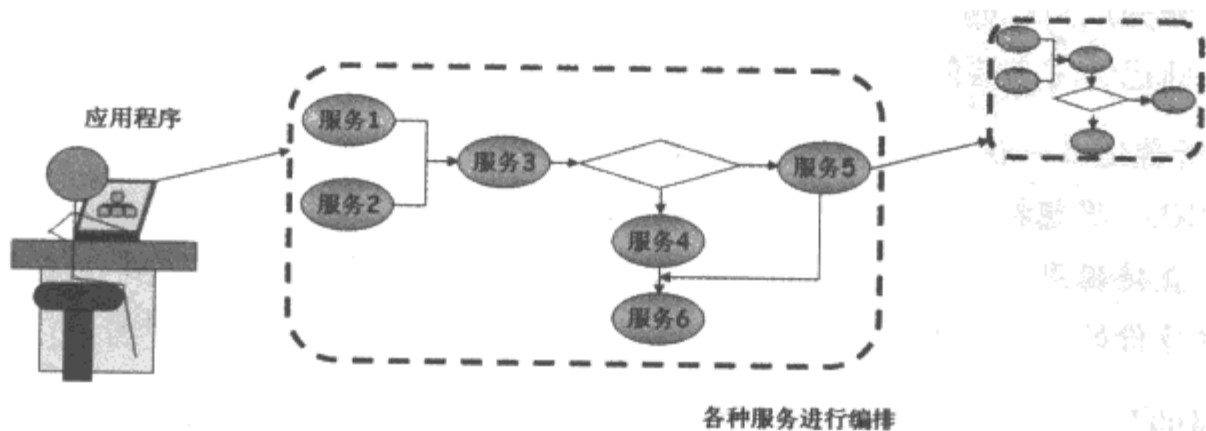


图 12-4 SOA 平台下的软件开发模式

在 SOA 下通过对服务进行编排，被封装完成后的服务对于需要使用这些服务的开发人员来

说, 就像调用本地方法那样去调用该服务的功能。这些服务既可以是原子服务, 又可以是由不同原子服务编排成的组合式服务。

最后总结一下, SOA 的特点如下。

- 抽象: 服务对外隐藏其他的细节内容, 只在服务自身的描述文档中提供需要遵守的通信约定。
- 松耦合: 服务和 service 之间具有最小的依赖性, 只需要保持相互之间的基本了解即可。
- 自治: 每个服务对自己内部的逻辑具有最高的指挥权。
- 重用性: 每个服务都实现了一种逻辑, 而这些逻辑是重用的, 可以用于解决其他问题。

“师兄, SOA 的思想这么好, 为什么以前没流行起来呢?”

“SOA 之所以沉寂至今才爆发, 原因就是当年并没有很好的技术去实现 SOA 的思想, 缺乏有力的技术支持, 它只好在沉睡中默默等待。”

“那是什么唤醒了 SOA 这头沉睡的大象呢?”

“唤醒 SOA 并将其推向风口浪尖的正是 Web Service 技术, 目前实现 SOA 有多种技术选择, 但是最好的也是最被广泛认可的解决方案就是 Web Service。”

写了 Web Service 就叫 SOA 吗? 不对, Web Service 只是技术支持, 要用 Web Service 这种技术实现 SOA 的思想才行, 就像要借助 Java 实现面向对象的思想一样。

根据以上的探讨可以得出一个结论: SOA 不是一种技术, 而是一种解决问题的思想。不管是客户与软件供应商, 还是软件供应商之间, 或是企业的内部系统, 都可以采用 SOA 的架构思想去处理问题, 大大提高系统间互联互通的能力。

### 12.1.2 零距离接触 Web Service 开发

前面小节的最后提到了 Web Service 是实现 SOA 思想的最好方式。本小节就来向读者介绍 Web Service 的相关知识, 并使用 Java SE 6.0 自带的工具包演示一个简单的 Web Service 示例, 为后续用 Web Service 实现 SOA 思想打下基础。

“师兄, 你刚才说的 Web Service 是什么啊? 是普通的 Web 应用吗?”

“当然不是啦, 这两种东西之间在技术上可是不太相干的啊。就像 Java 和 JavaScript, 二者只是名字上沾点边, 但是却没有很直接的血缘关系。”

“哦, 那总不是抽签决定的名字吧?”

“说得通俗一点, Web Service 应该算是这种技术的品牌名称吧, 酒香也怕巷子深, 如果不取个听起来比较火爆的名字, Web Service 还是很难流行起来的。”

其实, Web 应用和 Web Service 在实现思想上也有一些共同点, Web 应用是基于 HTTP 的, 这个通用的协议实现了用户与机器之间无阻碍的互通, 即只要是安装了浏览器, 所有机器提供的基于 HTTP 的信息, 用户都能够访问。

而 Web Service 实现的则是机器与机器之间的互通, 即只要是基于 Web Service 提供服务的机器, 都可以非常方便地进行服务功能的互相访问, 这就相当于在机器之间搭建了一个通用的通信平台。如通过 Web Service, .NET 平台和 Java EE 平台可以方便地实现互通。

Web Service 在电子商务的开发中有很大的优势, 同时可以为用户提供更高层次、更方便的资源共享, 如分布式计算等。从本质上来讲, 一个 Web Service 可以看作一个能被外界调用的方法,

直接用代码开发出来的 Web Service 一般都是原子服务。

“师兄，从 SOA 思想讲到 Web Service，这么多的理论，好压抑啊，来点示例调节一下气氛吧！”

“就知道你快听不下去了，好吧，我就来给你举一个在 Java SE 6.0 中开发简单的 Web Service 的示例。”

“必须得是 6.0 版本吗？其他版本无法实现吗？”

“以前，开发 Web Service 是很麻烦的，如果不用工具的话，半天能搞定就不错。因此用的人也就比较少，Java SE 6.0 时代的到来，才终结了这些烦琐和不便。”

从 Java SE 6.0 开始，Java SE 提供了支持 Web Service 开发的工具包，基于它可以快速开发出 Web Service，给开发部人员的感觉就像是写了一个普通的方法。下面给出了使用 Java SE 6.0 的工具包开发一个比较简单的 Web Service 的示例，该示例中 Web Service 的功能为接收用户的姓名输入并返回当日该用户的幸运指数，步骤如下。

## 1. 创建 Web Service 实现类

新建一个 Java 类，其名称为 MyLuckTestWebService，并写入如下代码。

```
1 package wyf;
2 import javax.jws.*; import javax.xml.ws.*; //引入相关包
3 @WebService //注解,通知系统所创建的类是会作为一个 Web Service 发布的
4 public class MyLuckTestWebService{ //定义提供 Web Service 的类
5     public String callLuck(String customer){ //定义 Web Service 中的服务方法
6         return customer + ",您今日的幸运指数为: " + (int)(Math.random()*100);
7     }
8     public static void main(String args[]){ //发布 Web Service
9
10        Endpoint.publish("http://localhost:8080/MyWebServices/MyLuckTestWebService"
11                           ,new MyLuckTestWebService());
12                           //指定 URL 和服务实现类并发布
13        System.out.println("Web Service 已经成功启动,等待访问...");
14    }
15 }
```

## 2. 编译并生成 Web Service

编译开发好的 MyLuckTestWeb Service.java 源文件，编译成功后打开命令行窗口，将当前路径调整至 Web Service 类所在的路径，输入“ws-gen -cp . wyf.MyLuckTestWebService”。该命令将对 Web Service 类自动进行包装，生成所需要的辅助类，执行该命令后的类文件目录如图 12-5 所示。

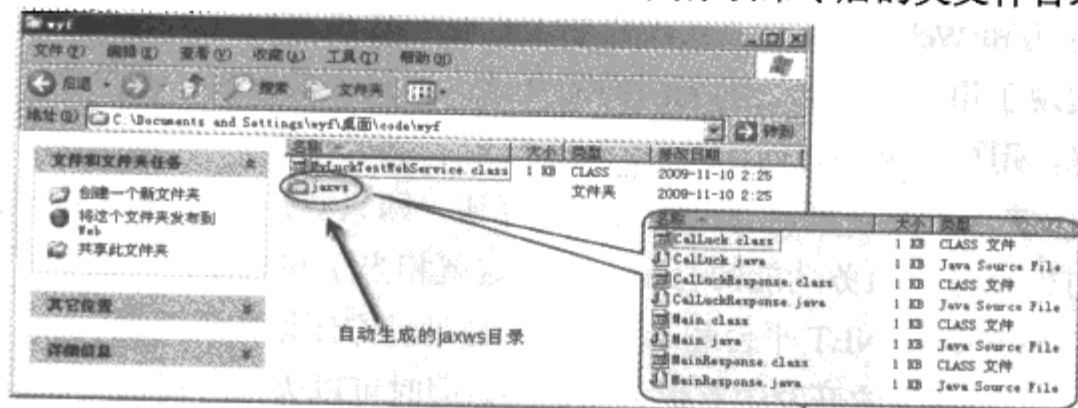


图 12-5 自动生成的 jaxws 目录

### 3. 发布 Web Service

运行 MyLuckTestWebService 类以发布 Web Service，运行成功后的效果如图 12-6 所示。



图 12-6 Web Service 运行时效果

### 4. 开发客户端辅助类

在开发客户端之前，需要首先为客户端生成一些访问对应 Web Service 的辅助类。这些辅助类的产生需要使用 wsimport 工具。假设客户端的目录为“c:\client”，在命令行窗口中输入“wsimport http://localhost:8080/MyWebServices/MyLuckTestWebService?wsdl”命令，将在 client 目录下创建一个辅助类的目录，如图 12-7 所示。

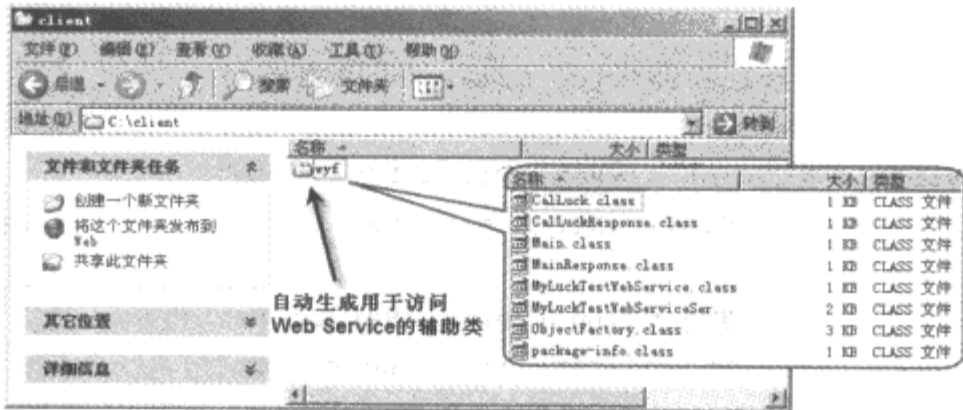


图 12-7 使用 wsimport 工具生成辅助类

这个 wyf 目录中包含的就是 wsimport 为访问 Web Service 而生成的辅助类。这里有必要提一下使用 wsimport 命令时输入的“http://localhost:8080/MyWebServices/MyLuckTestWebService?wsdl”，这个是 Web Service 的 WSDL 文件的 URL 地址。

每个 Web Service 在使用 wsgen 生成并发布后都会产生一个 WSDL 文件，这个 XML 文件告诉外界该 Web Service 都有什么样的功能，如有什么方法及方法输入参数、输出参数等信息。根据 WSDL，就能编写调用 Web Service 的客户端。

打开浏览器，在地址栏输入 Web Service 的 URL 并在其后添加“?wsdl”即可访问 WSDL 文件，如图 12-8 所示。

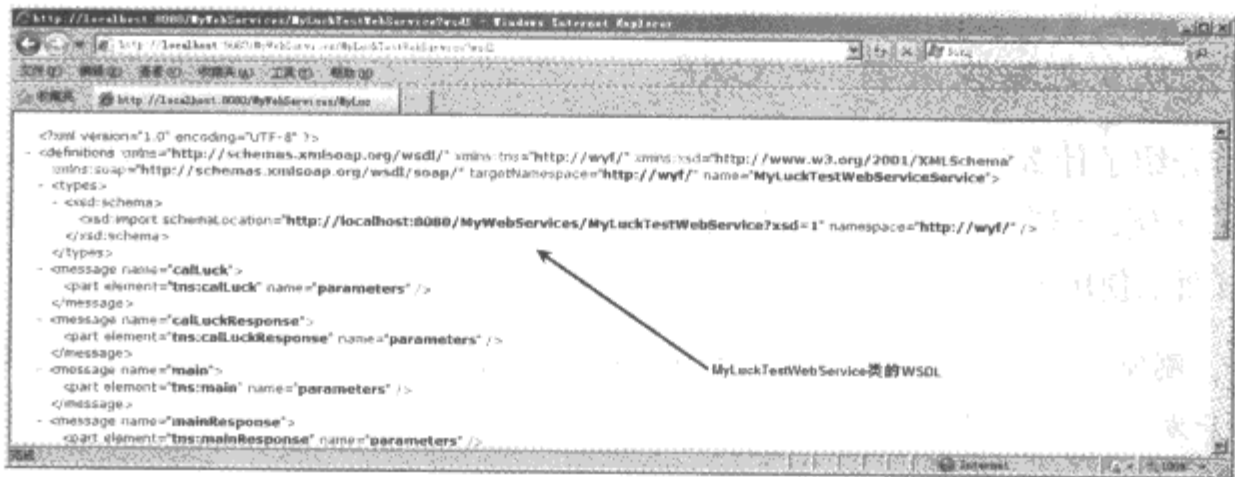


图 12-8 Web Service 的 WSDL



**提示** 前面用到的“wsген”、“wsimport”命令都在 Java SE 6.0 安装目录下的 bin 子目录中。

## 5. 开发客户端代码

生成辅助类之后，就可以开发客户端的代码了。在 client 目录下新建一个 Java 文件，取名为 MyLuckTest\_Client.java，并输入如下代码。

```

1  package wyf;
2  import javax.jws.*; import javax.xml.ws.*;          //引入相关包
4  import java.net.*;import javax.xml.namespace.*;    //引入相关包
6  public class MyLuckTest_Client{                    //声明访问 Web Service 的客户端类
7      public static void main(String args[]) throws Exception{
8          URL wsURL=new URL("http://localhost:8080/MyWebServices/MyLuckTestWeb
Service? wsdl");
9          QName qn=new QName( "http://wyf/", "MyLuckTestWebServiceService");
                                                    //服务名称
13         Service service=Service.create(wsURL,qn);    //使用动态服务
15         MyLuckTestWebService wsemPort=service.getPort(MyLuckTestWebService.class);
16         System.out.println(wsemPort.callLuck("Tom")); //输出访问结果
18         MyLuckTestWebServiceService wsems=new MyLuckTestWebServiceService();
                                                    //使用静态服务
19         wsemPort=wsems.getMyLuckTestWebServicePort();
20         System.out.println(wsemPort.callLuck("Tom")); //输出访问结果
21     }}

```

## 6. 编译运行客户端

客户端代码开发完成后，将其编译并运行，其输出结果如图 12-9 所示。

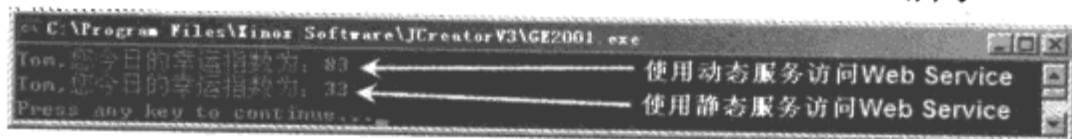


图 12-9 客户端访问 Web Service 示意图

由上述过程可以看出使用 Java SE 6.0 的工具包大大简化了 Web Service 的开发，通过使用程序注解，让开发服务实现类的服务方法和开发普通的方法基本一样。这使得开发人员可以将更多的精力放在业务逻辑上，不必再为原来 Web Service 开发的烦琐过程而感到 boring 了。

### 12.1.3 博采众长之集大成者——CXF

前一小节介绍了什么是 Web Service，并演示了一个使用 Java SE 6.0 自带的工具包开发简单 Web Service 的示例。但在实际开发中，Java SE 6.0 的工具包并不能很好地满足实际需求，因此本小节将会对市面上使用的主流开源 Web Service 开发框架做一个简单的介绍。

“蔡佳娃，刚刚介绍了如何使用 Java SE 6.0 的工具包来开发 Web Service，不过你可不要以为目前 SOA 的开发人员都是用这个啊！”

“啊，那都用什么啊？”

“目前市面上比较流行的 Web Service 开发框架有两个：CXF 和 Axis 2，我们首先研究研究 CXF。”

CXF 是一个免费的 Java SOAP 框架，其前身为 Codehaus 的 XFire。CXF 的核心是使用轻量级的 StAX 消息处理模型来对往来的 SOAP 消息进行处理的，StAX (Streaming API for XML) 是 Java EE 的一种技术标准。CXF 支持多种 XML 和对象之间的绑定机制以及 XML 消息的传输机制。

开发中可以将 CXF 集成到很多不同的容器和其他框架中，如 CXF 在思想上和 Spring 框架有很多互通的地方，Spring 可以与其非常好地结合在一起。其他的可以与 CXF 集成的开发框架还有很多，如 Plexus、PicoContainer 等。

下面通过一个简单的小例子来演示如何使用 CXF 框架进行 Web Service 的开发，该 CXF 应用运行在 Tomcat 服务器中。在本例中，客户端提交两个人名，服务端返回二者的缘分指数，具体开发过程如下。

### 1. 准备工作

进行基于 CXF 框架的 Web Service 开发之前，需要做一些准备工作。首先需要从 CXF 的官方网站上下载 CXF 的压缩包，目前的最新版本为 2.2.4，本例将以 2.1 版本为例。下载完成后，将其解压缩到指定的目录下，如 C:\apache-cxf-2.1。

然后在 Tomcat 安装目录下新建 endorsed 目录，将 C:\apache-cxf-2.1\lib 目录下的 jaxb-impl-2.1.6.jar、jaxb-api-2.1.jar、jaxb-xjc-2.1.6.jar 复制到新建的 endorsed 目录下。接着在 Java SE 6.0 安装路径下的 jre 目录中的 lib 子目录（如 C:\Program Files\Java\jdk1.6.0\jre\lib）下新建 endorsed 目录，将 jaxb-impl-2.1.6.jar、jaxb-api-2.1.jar、jaxb-xjc-2.1.6.jar、jaxws-api-2.1-1.jar 复制到 endorsed 目录下。

### 2. 服务端的开发

在 CXF 框架中，每个 Web Service 都对应到一个 Tomcat 服务器上的 Web 应用中，因此每个 CXF 应用一般都组织成一个目录或 war 包放到 Tomcat 的 webapps 目录下，其结构如图 12-10 所示。

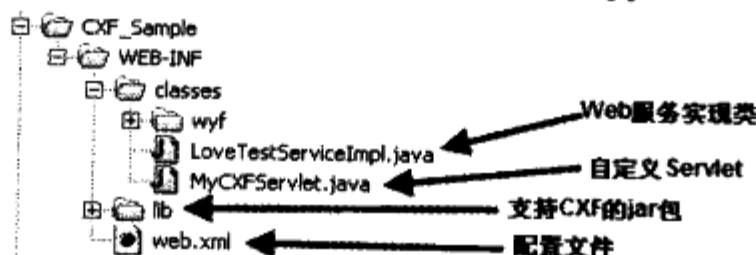


图 12-10 CXF 应用目录结构图

从图 12-10 中可以看出，CXF 应用目录的组织结构与传统的 Web 应用没有太大的区别。只是 classes 目录下多了自定义的用来发布 Web Service 的 Servlet 类文件，同时 lib 目录下多出了很多用来支持 CXF 的 jar 包。

与传统的 Web 应用一样，CXF 应用中也需要对自定义的 CXFServlet 进行配置。其配置信息同样写在 web.xml 配置文件中，其中配置此 Servlet 的部分代码如下所示。

```

1  <servlet>
2      <servlet-name>CXFServlet</servlet-name>           <!--Servlet 名称-->
3      <display-name>CXF Servlet</display-name>
4      <servlet-class>wyf.MyCXFServlet</servlet-class>    <!--Servlet 类名-->

```

```

5  </servlet>
6  <servlet-mapping>                                <!--配置 Servlet 映射-->
7      <servlet-name>CXFServlet</servlet-name>
8      <url-pattern>/services/*</url-pattern>
9  </servlet-mapping>

```

配置完了自定义的 CXFServlet，就该开发 Web Service 的实现类了。CXF 中的服务实现类一般就是应用了特定 Annotation（程序注解）的 POJO，下面给出了本例中服务实现类 LoveTestServiceImpl 的代码。

```

1  package wyf;
2  import java.util.*;import javax.jws.*;           //引入相关包
4  @WebService(name="LoveTestService",targetNamespace="http://bnkjs.com" )
                                           //使用程序注解
5  public class LoveTestServiceImpl{           //声明服务实现类
6      @WebMethod                               //使用程序注解
7      public String getLoveTest(String male,String female){ //定义服务方法
8          /*业务代码与本案例无直接关系，省去*/
9      }}

```

服务实现类开发完毕后，就可以开发发布该 Web Service 的代码了，在 CXF 中有两种发布 Web Service 的方式。本例中采用了直接继承 CXFNonSpringServlet 类，并重写其 loadBus 方法的方式，代码如下所示。

```

1  package wyf;
2  import javax.xml.ws.Endpoint; import org.apache.cxf.Bus; import org.apache.
cxf.BusFactory;
5  import org.apache.cxf.transport.servlet.*; import javax.servlet.*;
                                           //引入相关包
7  public class MyCXFServlet extends CXFNonSpringServlet{
                                           //继承自 CXFNonSpringServlet 类
8      @Override                               //使用程序注解
9      public void loadBus(ServletConfig servletConfig) throws ServletException{
                                           //重写 loadBus 方法
10         super.loadBus(servletConfig);
11         Bus bus = this.getBus();           //获取总线
12         BusFactory.setDefaultBus(bus);     //设置默认总线
13         Endpoint.publish("/LoveTestService", new LoveTestServiceImpl());
                                           //发布 Web 服务
14     }}

```

上述工作都完成后就可以部署并运行本应用了，启动 Tomcat 服务器，并在浏览器地址栏中输入：“http://localhost:8080/CXF\_Sample/services/LoveTestService?wsd/”。如果程序运行正常，在浏览器中可以看到如图 12-11 所示的 WSDL 文件。

### 3. 客户端的开发

开发客户端时首先要生成客户端辅助类，这个过程可以使用 Ant 工具来自动完成。首先在客户端目录下建立 gensrc 和 mysrc 两个目录，用来存放生成的辅助类源文件和编译后的类文件，Ant 的 build.xml 文件内容如下所示。

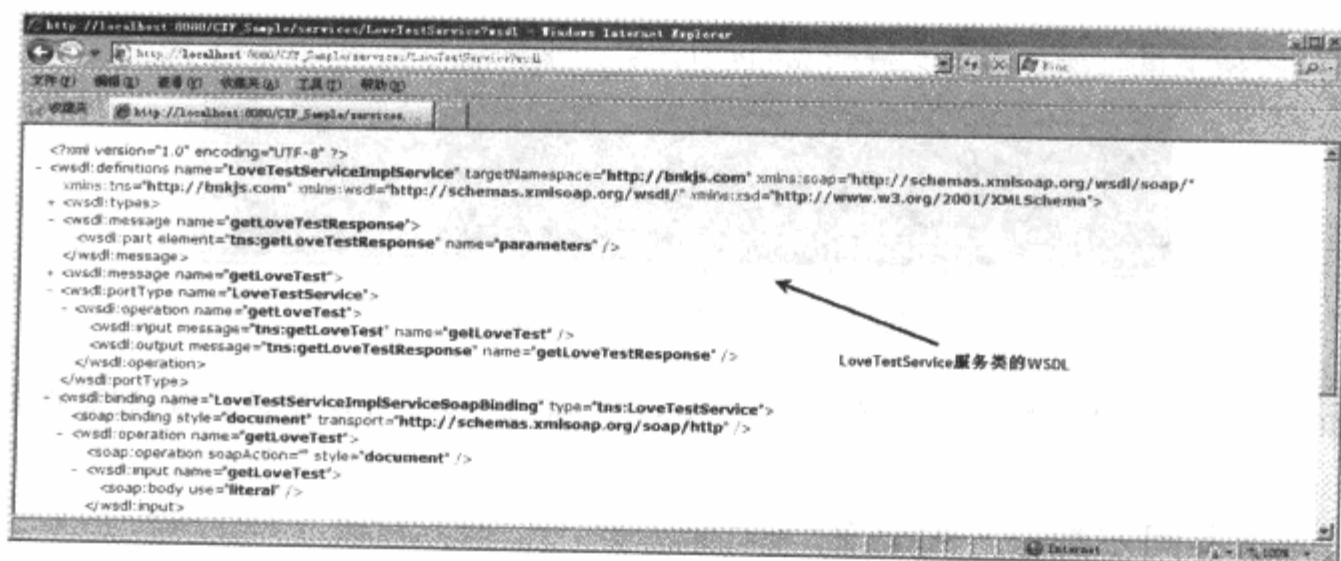


图 12-11 Web Service 的 WSDL

```

1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <project name="WYFCXF" default="compileStub" >
3      <target name="cxfwSDLToJava">
4          <java classname="org.apache.cxf.tools.wsdlto.WSDLToJava" fork="true">
5              <arg value="-client"/>          <!--只生成客户端辅助文件-->
6              <arg value="-d"/>              <!--指定目录-->
7              <arg value="gensrc"/>          <!--自动生成的源代码目录-->
8              <arg value="-p"/>              <!--指定辅助类所属包-->
9              <arg value="wyf"/>             <!--辅助类所属包名-->
10             <arg value="http://localhost:8080/CXF_Sample/services/LoveTest
Service?wsdl"/>
11         </java>
12     </target>
13     <target name="compileStub" depends="cxfwSDLToJava">
14         <javac destdir="mysrc" srcdir="gensrc/wyf">
15             <include name="*.java"/>      <!--编译自动生成的辅助类-->
16             </javac>                      <!--所有的 java 文件都要编译-->
17     </target>
18 </project>

```

生成辅助类之后就可以开发具体的客户端代码了，在 `mysrc` 目录下新建一个 Java 源文件 `Client.java`，并输入如下代码。

```

1  package wyf;
2  public class Client{
3      public static void main(String args[]) throws Exception{          //声明客户端类
4          LoveTestServiceImplService tsis=new LoveTestServiceImplService();
5          LoveTestService ts= tsis.getLoveTestServicePort();           //创建 Service 对象
6          String result=ts.getLoveTest(args[0],args[1]);                //获取功能 port
7          System.out.println(result);                                    //调用业务方法
8      }}                                                                    //打印输出结果

```

编译成功后就可以运行客户端了，由于需要传入入口参数，本例中可以使用 Ant 任务来运行客户端。客户端运行的时候需要许多 jar 包的支持，需要在运行的时候将这些 jar 包包含进

CLASSPATH 环境变量。在加入了支持的 jar 包后，客户端的运行结果如图 12-12 所示。

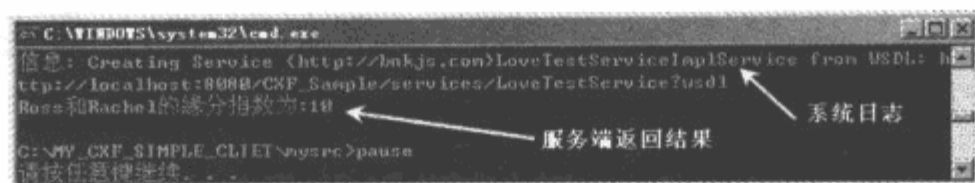


图 12-12 客户端运行结果图

**提示** 由于本书的篇幅所限，具体运行的 Ant 代码略去，读者可以参照 CXF 的资料自行完成。

#### 12.1.4 英雄不问岁数——Axis 2

“师兄，我觉得使用 CXF 开发 Web Service 的话，服务端虽然比较简单，但是还要经过一个 Servlet，有些绕远。客户端还算是比较简练，除了需要提前生成一些辅助类之外。”

“呵呵，你总结的还算正确。不过 CXF 通过程序注解+POJO 实现 Web Service 的开发，也算是非常不错优点了。”

“的确，没有程序注解估计开发起来肯定特要命，POJO 这个开发方式还真是不错啊！”

“看出来了吧，下面我们要介绍的 Axis 2 也采用了 POJO 的开发方式。同时 Axis 2 还有很多和 CXF 不同的地方，比如 Axis 2 就消除了你刚才说到的 CXF 中的 Web Service 要经过 Servlet 发布这个麻烦事，但同时它的客户端开发却要相对复杂些。”

Axis 2 是 Apache 的一个基于 Java 实现的 Web Service 框架，其中同时包含了服务器端与客户端的解决方案。Axis 2 是完全基于模块化方式实现的，这使得其非常容易在未来根据 Web Service 规范的发展添加新的功能模块。同 CXF 一样，Axis 2 也主张从 POJO 创建 Web Service。

下面将给出一个使用 Axis 2 开发 Web Service 的简单案例，该案例中客户端访问服务端，服务端将会返回天气预报信息。Web 服务部署在 Tomcat 服务器中，其过程如下。

##### 1. 部署 Axis 2 的 Web 应用

开发部署 Web Service 之前，首先要在 Web 容器中部署 Axis 的 Web 应用。首先从网上下载 Axis2-1.4-war.zip 文件，并对其进行解压缩，然后将解压缩后得到的 Axis2.war 文件部署到 Tomcat 下。部署完成后启动 Tomcat，在地址栏中输入“http://localhost:8080/axis2/”，若出现如图 12-13 所示的界面，则表示部署成功。



图 12-13 Axis 2 部署成功图



## 2. 开发服务端

每个 Axis 2 的 Web 服务都需要按照如图 12-14 所示的格式组织目录，并在开发完成后打包成一个 aar 包部署到 Axis 2 的 Web 应用中。从图中可以看出，一个 Axis 2 Web Service 的目录结构非常简单。

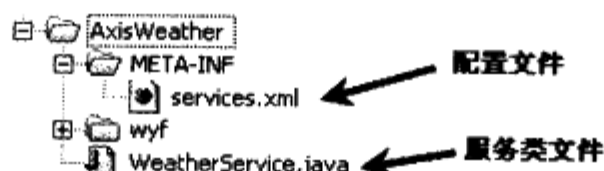


图 12-14 Axis 2 下的 Web 服务目录结构

目录结构中主要有两个组成部分，第一个是 META-INF 目录下的 services.xml 文件，其中包含了 Web 服务的配置信息。在 services.xml 文件中主要进行了消息的接收者、实现类、服务器名称等信息的说明，其代码如下所示。

```

1  <service name="WeatherService" scope="application"> <!--给出服务名称及生命周期-->
2      <description>Weather POJO Service</description> <!--给出服务描述-->
3      <messageReceivers> <!--消息接收者说明-->
4          <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
5              class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
6          <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
7              class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
8      </messageReceivers>
9      <parameter name="ServiceClass">wyf.WeatherService</parameter>
                                     <!--给出服务实现类-->
10 </service>

```

服务端另外一个需要开发的就服务实现类了，在 Axis 2 中，服务实现类都是基于 POJO 的，如本例中服务实现类 WeatherService 的代码如下所示。

```

1  package wyf;
2  import java.util.*;
3  public class WeatherService{                                     //服务实现类
4      public String[] zqk=new String[]{"晴","晴间多云","多云转晴","多云","阴","小雨","中雨","大雨"};
5      public String getForecast(String msgpre){
6          /* 业务方法具体代码省略 */
7      }}

```

开发完服务配置文件与服务实现类之后，就可以对 Web Service 进行打包部署了。首先将 AxisWeather 目录下的内容通过“jar -cvf AxisWeather.aar \*”命令打包到 AxisWeather.aar 文件中，然后通过 Axis 2 Web 应用的管理控制台将 AxisWeather.aar 进行部署。

在如图 12-15 所示的 Axis 2 欢迎界面下方点击 Administration 超链接，进入如图 12-16 所示的管理控制台登录页面，输入用户名“admin”和密码“axis2”，进入如图 12-17 所示的上传页面。

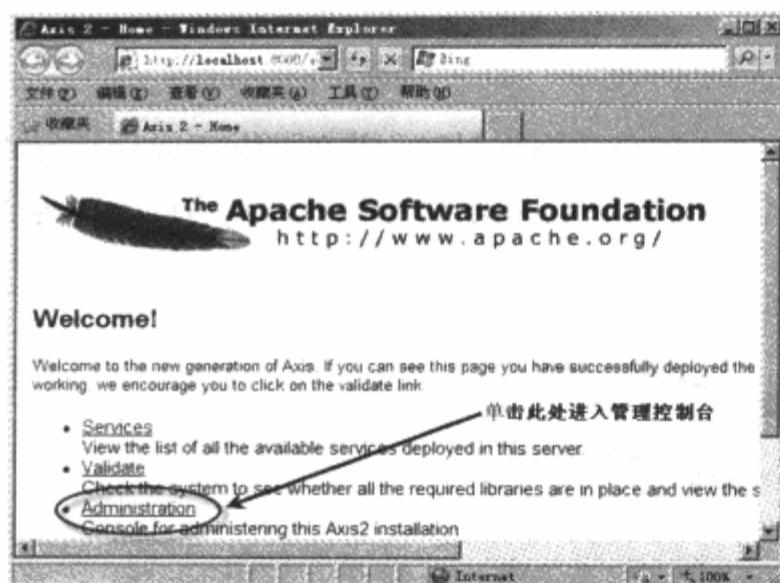


图 12-15 部署之进入管理控制台

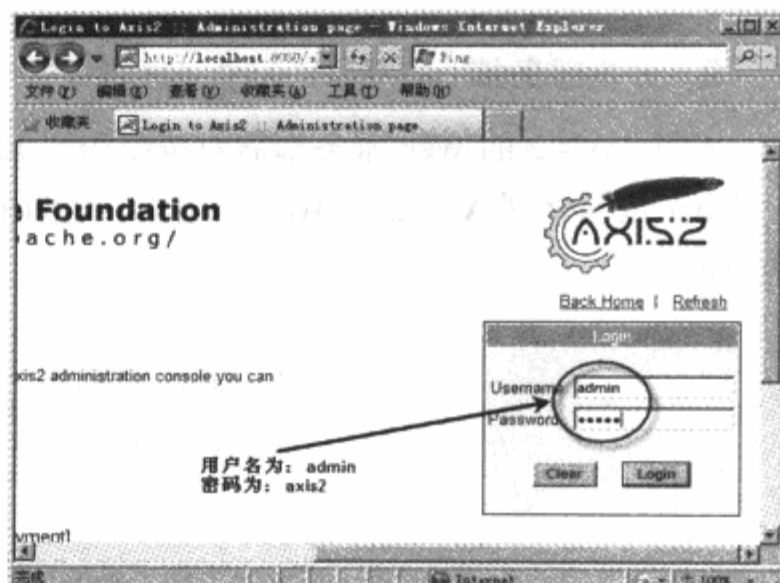


图 12-16 部署之输入账号密码

在上传页面中将打包好的 AxisWeather.aar 包上传到 Axis 2 的 Web 应用中。上传成功后，即可在 Available Services 中查看刚刚上传的 Axis 2 Web Service 了，如图 12-18 所示。



图 12-17 部署之上传打包文件

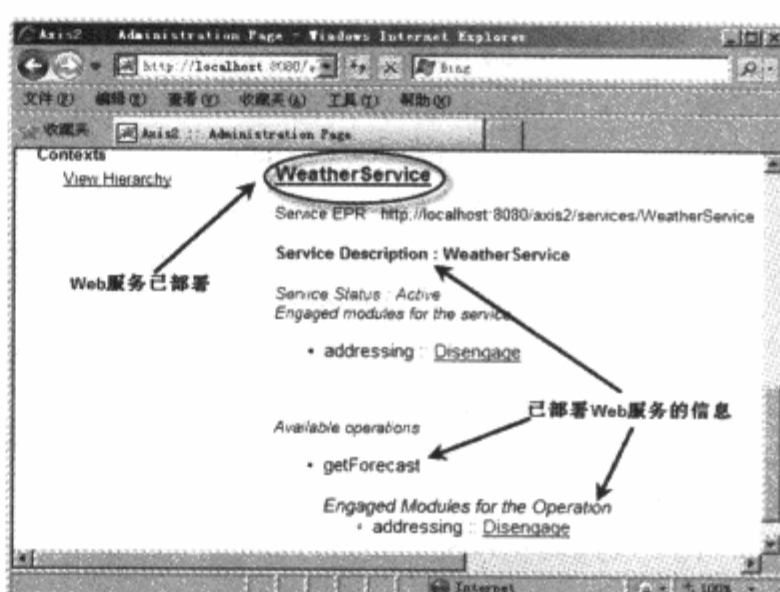


图 12-18 部署之查看已部署服务的信息

### 3. 开发客户端

服务端部署成功以后，就可以开发客户端代码了。客户端的开发相比服务端要稍微复杂一些，其代码如下所示。

```

1  import javax.xml.namespace.QName; import org.apache.axis2.*; //引入相关包
3  import org.apache.axis2.addressing.*; import org.apache.axis2.client.*;
                                     //引入相关包
5  import org.apache.axis2.rpc.client.*; import java.util.*; //引入相关包
7  public class SelfRPCServiceClient{
8      public static void main(String args[]){
9          try{
10             RPCServiceClient serviceClient = new RPCServiceClient();
                                     //创建 RPC 服务客户端
11             Options options = serviceClient.getOptions(); //获取操作列表
12             String serviceUrl="http://localhost:8080/axis2/services/Weather
Service";
                                     //服务的 URL

```

```

13      EndpointReference targetEPR = new EndpointReference(serviceUrl);
                                           //创建端点引用
14      options.setTo(targetEPR);
15      QName opGetForecast =new QName("http://wyf", "getForecast");
                                           //创建 QName
16      Object[] opGetForecastArgs = new Object[] { "wyf" };
                                           //创建参数值数组
17      Class[] returnTypes = new Class[] { String.class };
                                           //创建参数类型数组
18      Object[] response = serviceClient.invokeBlocking //调用服务
19      (opGetForecast,opGetForecastArgs,returnTypes);
20      String result=(String)response[0];
                                           //从服务端获取响应
21      if(result!=null){
22          /* 将服务端发回的数据打印输出 */
27      }}
28      catch(Exception e){ e.printStackTrace(); } //捕获可能抛出的异常
29  }}

```

客户端代码无论是编译还是运行，都需要 Axis 2 中一些 jar 包的支持。在 CLASSPATH 环境变量中引入编译和运行所需的 jar 包，对客户端代码进行编译并运行，结果如图 12-19 所示。

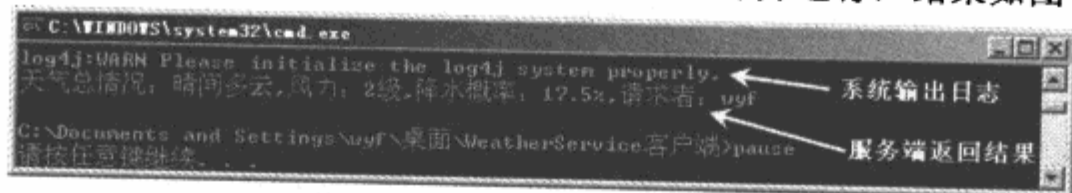


图 12-19 客户端运行结果图

通过比较可以看出，基于 POJO 开发的设计使得 Axis 2 服务端开发比较简单，尤其是 Web 服务的实现类。不过客户端的开发就稍微复杂一些了，但读者朋友们也不用担心，其实 Axis 2 也提供了类似 CXF 的用 Ant 自动生成辅助类的方式。如果采用这种方式进行客户端的开发，就会方便很多了，有兴趣的读者可以参照 Axis 2 的资料自行操作。

### 12.1.5 走近 ESB——企业服务总线

“师兄，听了你关于 Web Service 的介绍，尤其是那两个开源框架的讲解，让我对 Web Service 的敬仰有如滔滔江水连绵不绝啊！”

“呵呵，不过你不要忘了我一开始说过的，Web Service 只是实现 SOA 的一种技术，SOA 才是我们需要探讨的问题呢。”

“嗯，那我还留着一句‘又有如黄河泛滥，一发不可收拾’给 SOA 呢，呵呵。”

“哎，你可真够贫的，闲话少说，我们最后来提一下实现 SOA 思想各个环节中最重要的一个关键人物——企业服务总线（ESB）。首先来说说它的起源和特性。”

#### 1. ESB 的起源

介绍 ESB 之前有必要先提一下点对点集成，点对点集成是指某一种应用依赖于另一种特定的

应用。比如说一个 Web Service 开发出来并被公布，如果想要调用该服务功能，就必须依照其对外接口开发一个依赖于该 Web Service 的应用，才可以调用这个 Web Service。图 12-20 以一个财务系统和销售系统之间的服务依赖关系说明了这个问题。

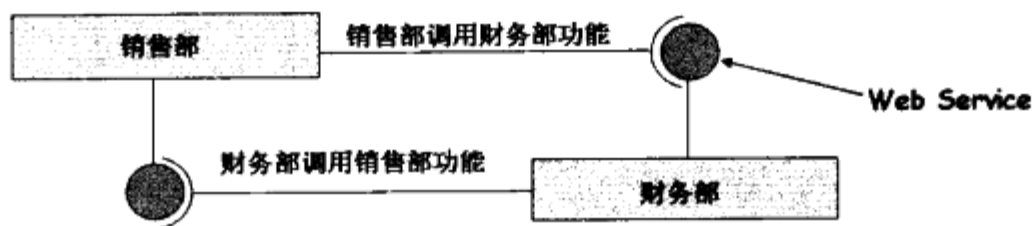


图 12-20 点对点集成模式图

点对点集成在模块之间产生了紧耦合，比如说公布了一个 Web Service，它有固定的 URL 和自己的服务实现接口，可以被其他的模块访问。但是如果这个 Web Service 的位置发生变化而改变了自己的 URL，或者修改了自己对外提供的 API，那么对于调用方来说，这个原有的服务就算是彻底作废了。

而前面强调过的之所以 SOA 不等同于 Web Service 的原因，就是因为 Web Service 只是 SOA 的技术支撑之一，只用其无法满足 SOA 所要求的松耦合。因此，开发出一个 Web Service 并不能代表做到了 SOA。

“师兄，我看出来了，点对点集成的确不是个好的集成方式。”

“不过你千万不要认为点对点是一无是处的，很多特定的场合必须使用点对点集成。虽然它的缺点是紧耦合，但它的好处就是快，有时如果对于速度的追求很高甚至都不会通过 Web Service 进行信息交换而是采用最原始的 Socket。”

“说得也是啊，Web Service 组件中对数据需要进行其他的转接，肯定会在效率上有所损失。”

系统点对点集成会带来诸如紧耦合等问题，为了解决这些缺陷，随着 SOA 的不断发展，就产生了通过 ESB（Enterprise Service Bus）进行集成的方式，其最主要的任务之一就是消灭紧耦合并实现松耦合。

在 ESB 集成方式中，松耦合主要体现在服务点位置透明和 Schema 转换上，这就使得功能的调用者不需要关心目标服务提供者物理位置的变化、接口的变化，总是按照既定的调用策略去实现功能调用即可。除此之外，ESB 还提供诸如服务聚合、强制安全、监控等功能，使得 Web 服务的管理更加简便高效。

## 2. ESB 剖析

“蔡佳娃，刚才谈的主要是为什么要选择 ESB 以及 ESB 优于别人的特性。感觉如何啊？”

“我觉得 ESB 的确能够非常好地诠释 SOA 思想，我对它的敬仰有如滔滔江水连绵不绝……”

“哎，你就不能换句吗？”

“呵呵，不过师兄，ESB 的这些特性是如何体现出来的呢？”

“嗯，这个问题还像点样，下面我们就来对 ESB 做个比较深入的研究。”

传统 Web Service 的调用是这样的：甲方开发出一种功能，将其暴露为服务点，当乙方需要调用这个功能时，就根据其服务接口进行编程直接调用，如图 12-21 所示。这种调用是依赖位置与接口的，而且由于双方直接连接，不利于监控，编排服务点也不太方便。

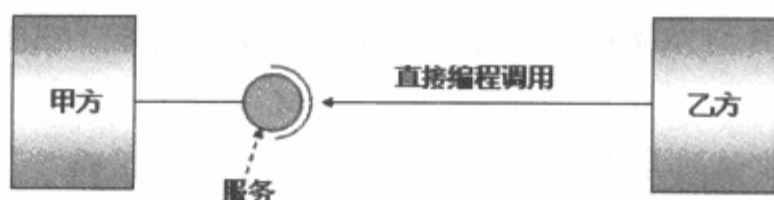


图 12-21 传统 Web Service 调用过程

而 ESB 的工作原理却是这样的：甲方开发出一个服务之后，将其注册到 ESB，ESB 将其包装并提供一个代理服务。当乙方需要进行功能调用时，不直接去找甲方，而是通过 ESB 中的代理服务进行调用。如果甲方的服务位置发生变化，那么只需要重新注册到代理服务，修改注册规则即可，乙方不受影响。ESB 缓冲层的作用可以很好地将变化限制在一定范围内，对外保持不变，如图 12-22 所示。

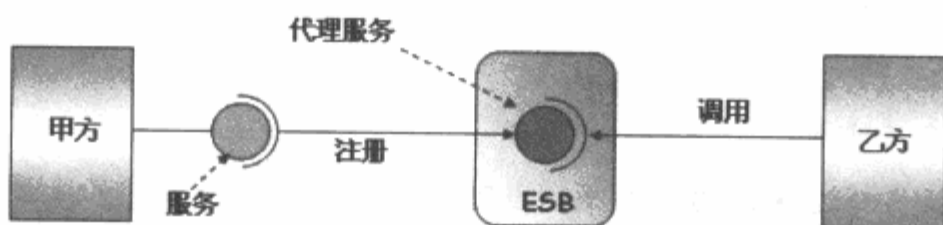


图 12-22 通过 ESB 改善紧耦合

再来讨论接口变化的问题，甲方注册到代理服务的时候，就将其接口映射到代理服务的接口，这时二者是相同的。当甲方的接口发生变化时，只需要修改甲方与代理服务接口间的映射规则，而代理服务对乙方所开放的接口是可以保持不变的。

进一步思考，ESB 还可以作为交通枢纽。假设在甲方和乙方的基础之上，又增加了一个丙方，丙方提供与甲方功能一样的服务，注册到相同的 ESB，这两个功能上服务是对等的。当乙方又来寻求功能调用时，不会直接去找甲方或丙方的代理服务，而是由 ESB 提供一个路由模块，如图 12-23 所示。

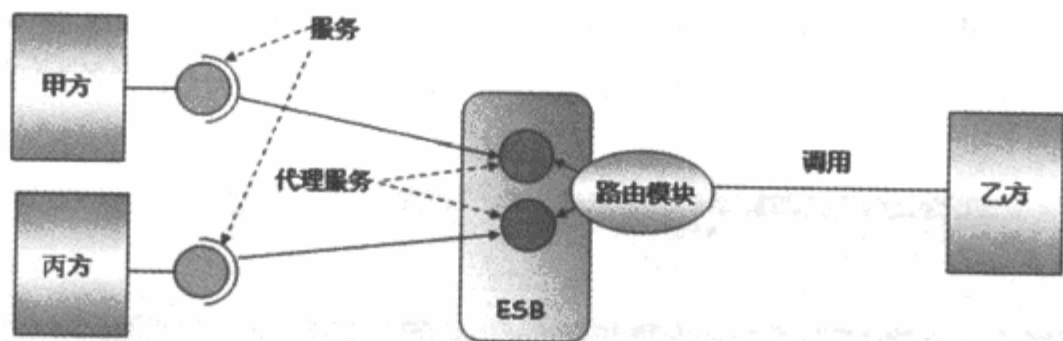


图 12-23 ESB 实现负载均衡

这样一来，当乙方的调用请求到来时，路由模块如果发现甲方的代理服务过于繁忙，而丙方相对清闲，就会将此次调用交给丙方的代理服务去做，这样可以很好地实现负载均衡，避免了拥堵现象。

ESB 中的路由模块除了作为交通枢纽，还可以作为导购员，根据不同的质量要求将功能调用请求路由到不同的代理服务。假设甲方提供的服务质量没有丙方好，当乙方对功能的调用有高质量要求时，就可以在调用规则中指出，ESB 的路由模块可以根据乙方对质量的要求，将调用请求转发给不同质量服务的代理服务。

“听师兄你这么一介绍，ESB 还真是 SOA 麾下的一员猛将啊！”



“那是当然，不过 ESB 可不是张飞那样只是能征善战的虎将，ESB 可是个能文能武像周瑜一般的儒将呢。”

“能文能武？如果刚才说的算是‘武’，那‘文’体现在哪里呢？”

除了上述优势，ESB 还可以运用在大型 Web Service 项目的开发中。大型的 Web Service 项目开发中，必然需要对原子服务进行编排。而如果没有 ESB，那么对服务点的编排只能留到工程的最后去做，这显然拖长了开发周期。而如果采用 ESB 技术，那么这个问题就迎刃而解了，方案如下。

每个开发团队在开发自己负责的服务点之前，先花很短的时间开发一个伪服务（Mock Service），开发完成后将其挂到 ESB 里面，这样负责服务点编排的小组就可以开工了。等真正的服务写好后，只要将 ESB 中代理服务点的映射规则修改一下即可，如此一来便实现了并行开发，大大缩短了工期。

“哇，听起来还真厉害啊。”

“是啊，我就曾经参与过这样的开发呢。当时是日本的一个项目，开发团队有 IBM 中国、IBM 日本和项目客户自己的工程师。当时我们就面临着很大的困难，一个是北京的团队如何在无法获得东京真实系统的条件下完成自己的测试，另一个就是如何使两地的团队互相不受影响各自开发。”

“哦，那你们是怎么解决的呢？”

“我们将系统分成若干个相互独立的服务，分派给各个团队，并且首先要求每个开发人员提供一个简单的‘伪’实现。各个服务的开发小组进行自己开发的同时，服务流程开发人员也在同一时间进行服务的编排。这种灵活的特性使得我们的开发过程非常流畅，简直就是享受。”

“哇塞，看来我对 ESB 的敬仰，应该有如 Pacific Ocean 般的无边无垠了。”

目前，市面上能提供 ESB 实现的平台中比较著名的是 Oracle（前 BEA）发布的 Oracle Service Bus（OSB，原名 AquaLogic Service Bus），OSB 能运行在 Windows、Linux 和 Solaris 系统上，它提供了现代企业应用所需要的如前所述的松耦合、位置透明、服务中介等各种功能。OSB 的控制台如图 12-24 所示。

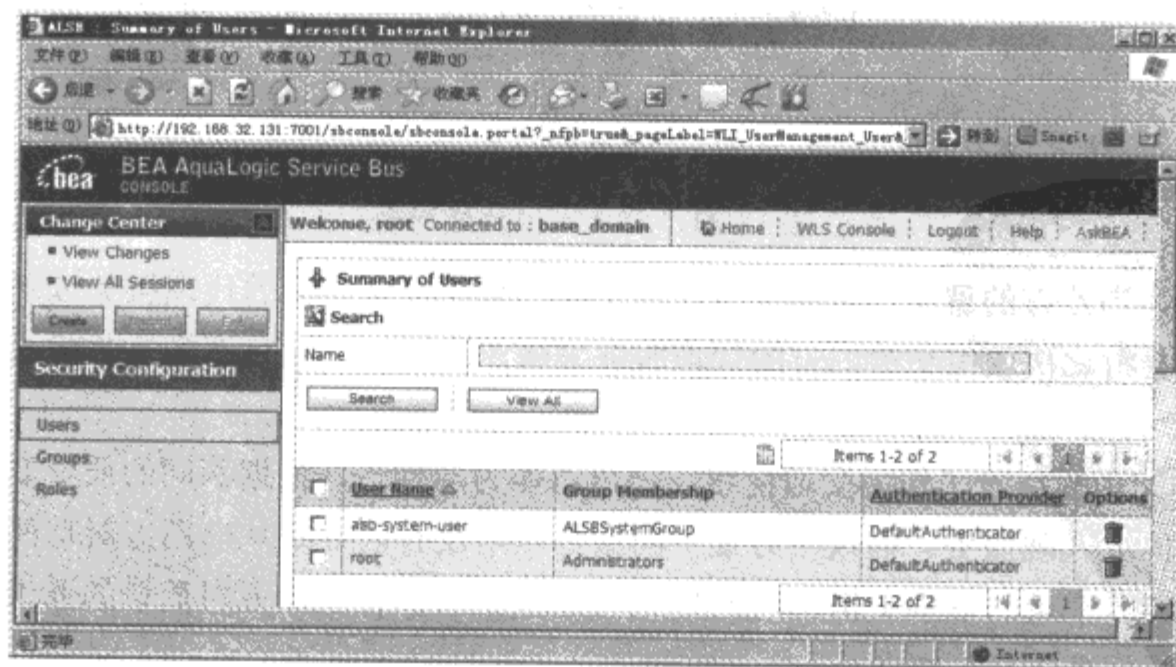


图 12-24 Oracle Service Bus 的管理控制台

看到 Oracle 的作品，一般读者都会联想到一个字——“贵”。但读者朋友们不用担心，ESB 的实现不但有昂贵的 OSB，也有很多免费的开源项目，OpenESB 就是其中比较有名的一个，有兴趣的读者可以下载下来研究研究。

## 12.2 富客户端应用（RIA）

十年前，软件开发中谈论最多的或许是如何实现某种复杂的业务逻辑以满足客户的要求，而现在随着软件开发技术越来越成熟，另一个因素越来越受到开发人员的重视，那就是用户体验。在“用户体验”这个新词的背后，是同样新奇的富客户端应用，本节就来讨论一下 Web 开发中的富客户端（RIA）。

### 12.2.1 从平淡到酷炫——RIA 与 AJAX

“师兄，你太狠了，说讲新技术，一上来就给我整 SOA，这么油腻的东西我哪能消化得了啊，不行，你得陪我肠胃损伤费、精神损失费、误工费等。”

“呵呵，好好，我的错，下面师兄我给你整点清淡的菜，咱们来谈谈 Web 开发中的富客户端应用 RIA。首先，咱们看看 RIA 的发展过程。”

Google 的 Gmail 推出之前，Web 页面一直以朴素实用方便着用户，但是 Gmail 颠覆了这个状况，在 Web 开发界掀起了一股 RIA 的风潮。RIA 是 Rich Internet Application 的缩写，由于其对于 Web 开发带来了席卷式的革命，很多人都将 RIA 称作是 Web 2.0 的代名词。

RIA 技术打破了传统 Web 页面死板、不灵活的旧模式，换之以更加丰富的内容、更加先进的信息交互方式，以及更加酷炫的页面效果。图 12-25 和图 12-26 是两个采用 RIA 技术的 Web 网站的页面，从中读者朋友们应该可以看出一些端倪。



图 12-25 RIA 应用实例 1



图 12-26 RIA 应用实例 2

“师兄，你早就该讲 RIA 嘛，这个听起来多轻松啊！看着就秀色可餐。”

“呵呵，不过身为开发人员，仅仅满足这些酷炫的界面可不够，还要研究这些效果是如何制造出来的才行。”

“嗯，那这些美味佳肴是如何烹饪出来的呢？”

“RIA 的实现技术很多，主要有 AJAX、Flash、JavaFX 等。其中主要还是 AJAX 技术，刚才提到的 Gmail 就是采用的 AJAX 技术来征服用户的。因此，我们就先来初探一下 AJAX 的发展史。”

AJAX 是 Asynchronous JavaScript and XML 的缩写，虽然“AJAX”这个名字直到 2005 年才被 Jesse James Garrett 首次提出，但 AJAX 技术却可以追溯到 20 世纪 90 年代末。AJAX 的发展史如图 12-27 所示。

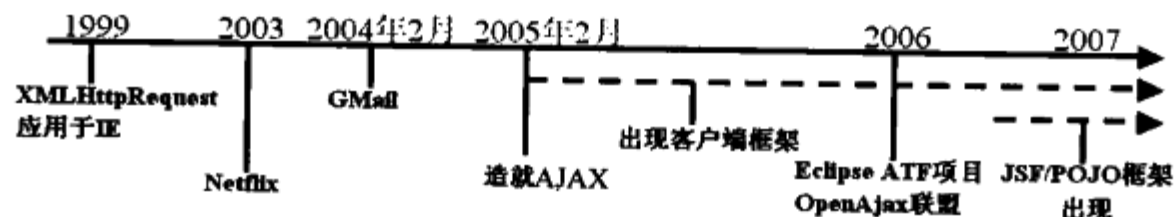


图 12-27 AJAX 发展史

1999 年 XMLHttpRequest 技术首次应用于 IE，标志着客户端通过脚本与服务器通信时代的开始，随后 AJAX 在 Netflix（在线影片租赁商）和 Gmail 上的成功运用使得 AJAX 迅速走红并发展壮大。现在，AJAX 技术已经非常成熟，并出现了诸如 DWR、Dojo 之类的客户端框架。

AJAX 并不是一项新的技术，而是对多项技术进行的一种全新的组合使用方式。这些技术包括 JavaScript、XHTML、CSS、DOM、XML、XMLHttpRequest 等。各种技术在 AJAX 引擎中的作用如图 12-28 所示。

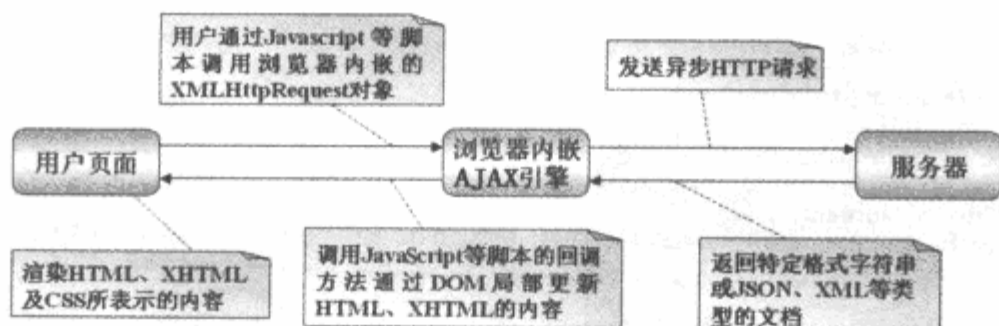


图 12-28 不同技术在 AJAX 中的作用

传统的 Web 应用遵循“请求-响应”的同步交互过程，这使得浏览器或者在向服务器提交请求，或者在等待服务器处理自己的请求，而后者往往居多。同步的信息交互模式使得浏览器在等待的时间里无事可做，页面也静如死水，对于客户来说，这是非常糟糕的体验。

而 AJAX 由于在用户与服务器之间引入了一个中间媒介——AJAX 引擎，从而消除了网络交互过程中单纯的“处理-等待”现象。AJAX 引擎主要负责更新用户界面以及与服务器之间进行信息交互，其最大的特性就是采用了异步信息交互，不会由于用户的一次请求而陷入长时间沉寂的等待。

AJAX 技术可以为 Web 应用带来一些新的特性，如不需要刷新页面即可实现网页内容的实时更新、不需要跳转页面即可与服务器进行数据交互等。除此之外，AJAX 还有很多优点，如可以减轻服务器负担、可以在多种平台下很好地应用、提供较强的安全性等。

“虽然 AJAX 给我们 Web 开发界带来了一场革命，但是 AJAX 也有其自身的缺陷。由于 AJAX 是一系列技术的大融合，所以不可避免地将所有技术的缺陷集中到了一起。”

“师兄，那我说一个你看算不算缺点，据我浅薄的经验来看，开发一个不算复杂的 AJAX 应用，JavaScript 的代码要写好多好多，非常累人。”

“呵呵，你这个算是个缺点吧，不过现在这个缺点已经基本不存在了，目前市面上有很多基于 AJAX 的开发框架，如 DWR 和 Dojo。”

DWR (Direct Web Remoting) 是一个开源类库，由 Getahead 公司开发。在 DWR 框架中主要包含两个部分，一个是运行在客户端浏览器的 JavaScript，这部分负责与服务端通信和更新页面内容；第二个部分是运行在服务端的 Servlet，这部分用来处理请求和返回响应结果。

传统的 AJAX 应用开发需要对服务端的 Java 代码和客户端的 JavaScript 代码分别包装以进行通信，这使得开发变得很烦琐。而 DWR 则免去了开发人员这方面的苦差事，DWR 会在后台根据 Java 类代码自动生成客户端 JavaScript 代码，并对客户端和服务端的通信内容自动进行包装。其交互过程如图 12-29 所示。

DWR 通过对每个服务端的 Java 类（通常为 POJO）动态生成包含在 Web 页面中的相应 JavaScript，使得在客户端浏览器中能够像调用本地代码一样调用 Web 服务端 POJO 中的业务方法。图 12-29 中浏览器所调用的 getHobbies 方法其实就是服务端 HobbiesBean 中的 getHobbies 方法。

使用 DWR 可以在很大程度上简化 AJAX 应用开发的难度和工作量，因为 DWR 能够实现类似 RPC 的远程功能调用，所以比较适合做业务功能的开发。



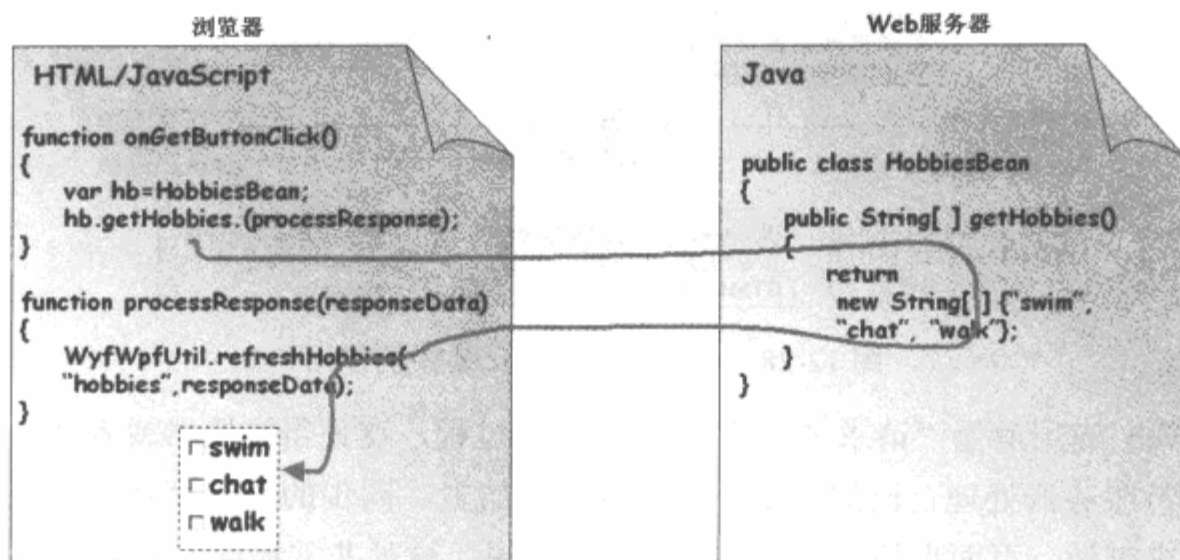


图 12-29 DWR 的交互过程

### 12.2.2 酷炫背后的基石——核心 JavaScript

“师兄，AJAX 真是不错，而且这东西开发出来直接拿到浏览器上就能部署，比钻研 SOA 方便多了，我回去得抓紧学习学习。”

“是啊，不过 AJAX 涉及的技术有好多呢，你准备先学哪个啊？”

“这回我不说了，估计我说出来也不对，师兄你还是直接告诉我吧。”

“呵呵，你好没自信啊！好吧我来说，想要驾驭好 AJAX，使其产生烟花般的绚丽效果，就必须研究烟花内部的成分——JavaScript。”

JavaScript 在 AJAX 技术中的作用就像黏结剂，它将其他的技术绑定在一起，所以说 AJAX 的应用中几乎全都是 JavaScript 代码。但是由于 JavaScript 是一门起点较低的脚本语言，很多对 JavaScript 并没有做深入学习的开发人员直接从事 Ajax 应用的开发会很吃力。

就像核心 Java 是 Java 开发人员必不可少的基石一样，核心 JavaScript 也是脚本开发人员出师之前的必修课，本小节将对核心 JavaScript 做一个简单的介绍。

## 1. JavaScript 的对象

JavaScript 虽不是面向对象编程，但也是基于对象的。因此在开发中可以方便地创建自己的对象，其简略语法如下所示。

1 <引用名>={<属性名>:<属性值>,[<属性名>:<属性值>]};

与 Java 中的语法不同，JavaScript 在声明的时候就顺便将创建的工作也完成了。在 JavaScript 中，对象实际上可以看做是数组，其成员不仅可以用“<引用>.<属性>”的方式来访问，还可以用“<引用>[<属性>]”的方式来访问。如下面代码中的第 3 行和第 4 行输出的结果是相同的。

[illegible]



```
5    </script>
```

“蔡佳娃，你能看出来这种创建对象的方式有什么缺点吗？”

“我想想啊，这种方式将声明和创建放到了一起，不利于重用吧。比如我要想再创建一个类似的对象，就得按照同样的格式再写一次，很不方便。”

“非常正确，下面我们就来看另一种创建对象的方式。”

在 JavaScript 中也可以像 Java 那样先声明类，再创建对象，其基本语法如下所示。

```
1    function <类名>([构造方法参数列表])    {
2        this.<属性名 1>=<构造方法参数 1>;    //对属性的声明和初始化
3        .....
4        this.<属性名 n>=<构造方法参数 n>;
5        this.<方法名 1>=<方法名 1>;          //挂接成员方法
6        .....                               //方法可以是在任意地方声明的方法
7        this.<方法名 n>=<方法名 n>;
8    }
```

上述代码中定义对象属性的同时也对其进行了初始化，而对于成员方法的声明，其实就是把已经写好的方法分配给该类作为一个成员而已。下面给出一个先声明类后创建对象的 JavaScript 示例，其代码如下所示。

```
1    <script language="JavaScript">
2        function Student(sno,sname,sage){    //声明 student 类
3            this.sno=sno;                    //初始化学号属性
4            this.sname=sname;                //初始化学生名字属性
5            this.sage=sage;                  //初始化年龄属性
6            this.toString=toString;
7            //把外面的 toString 方法挂接到 Student 中的 toString 成员
8        }
9        function toString(){                //声明 toString 方法
10           var result="学号: "+this.sno+"<br>"+"姓名: "+this.sname+"<br>"+"年龄: "+this.sage+"<br>";
11           return result;
12       }
13       var tom=new Student("10058","赵云",28);    //创建 Student 对象
14       document.write(tom.toString());            //调用 Student 对象的 toString 方法
15   </script>
```

如需要对 JavaScript 中的对象属性进行遍历，可采用 for-in 语句。例如对上面代码中的 tom 对象属性进行遍历时，可以使用如下代码完成。

```
1    for(var i in tom){
2        document.write(i+": "+tom[i]+"<br>");    //此处可用具体业务代码替代
3    }
```

需要注意的是，在 JavaScript 中可没有像 Java 中垃圾收集器那样的工具来自动管理内存，所以不用的对象要及时删除，释放对象时应使用“delete”操作符。

## 2. 原型和注射

“哇, JavaScript 虽然是个弱类型的语言, 不过能创建对象还真是个很好的特性呢, 不错不错。”

“在 JavaScript 中不仅可以创建新对象, 还可以扩展原有的对象呢。如果你对哪个 JavaScript 的内置对象不满意, 就可以进行自定义的补充和修改。”

“真有这么方便吗? 师兄你讲给我听吧。”

JavaScript 支持原型 (prototype), 通过使用原型可以向已有的对象类型注射新的方法和属性, 对对象的功能进行扩展, 同时也可以覆盖原来对象的方法。下面给出一个对 JavaScript 内置的 Date 对象进行扩展的示例, 代码如下所示。

```

1  <script language="JavaScript">
2      function toWyfString(){                                //定义新方法
3          var ss=this.wyfTime+this.toGMTString();
4          return ss;
5      }
6      function toGMTString(){                                //定义覆盖方法
7          return "HaHa!!!我把 Date 对象原来的 toGMTString 方法覆盖了! ";
8      }
9      Date.prototype.wyfTime="WYF_TIME: ";                  //将新属性注射到 Date 对象中
10     Date.prototype.toWyfString = toWyfString;              //将新方法注射到 Date 对象中
11     d=new Date();
12     document.write("扩展的 toWyfString 新方法: "+d.toWyfString()+"<hr>");
13     document.write("Date 对象的 toGMTString 方法: "+d.toGMTString()+"<hr>");
14     Date.prototype.toGMTString = toGMTString;              //覆盖 Date 对象原有的方法
15     document.write("被覆盖后的 toGMTString 方法: "+d.toGMTString()+"<hr>");
16 </script>

```

以上代码的运行结果如图 12-30 所示。

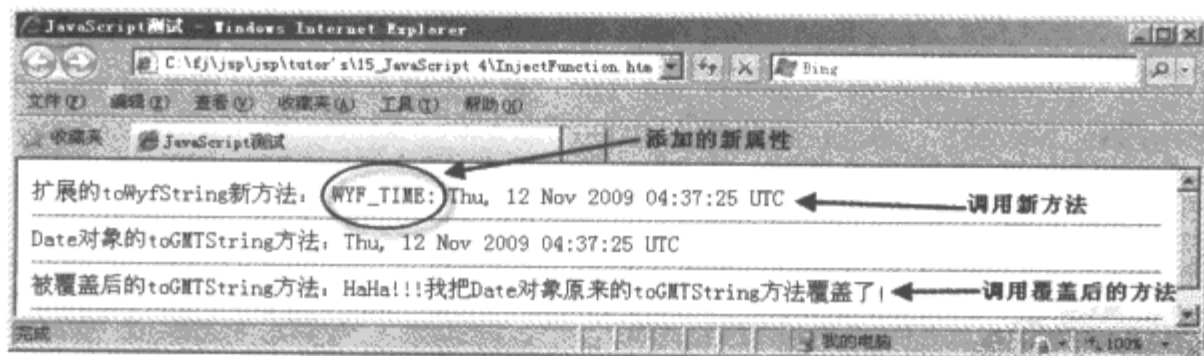


图 12-30 JavaScript 使用原型示例

JavaScript 支持原型的特性可以使开发变得更加灵活, 如 JavaScript 的 String 对象没有 trim 方法, 当开发中有很多对字符串的这种去空格处理时, 就可以考虑为其注射一个 trim 方法。

“JavaScript 还真是灵活啊。”

“不仅如此呢, JavaScript 还可以嵌入执行脚本, 把字符串作为 JavaScript 脚本来执行。例如下面这句代码。”

```

1  eval("alert('Hello World!')");

```

“在这句代码中，eval 方法将传入的字符串作为 JavaScript 的脚本语句来执行，这样就可以在运行过程中动态生成 JavaScript 语句，大大提高了程序的灵活性。”

“师兄，JavaScript 本来就是动态的了，还可以动态生成 JavaScript 语句，真是灵活啊！”

灵活的语言并不一定是简单的，或许需要更加留神以防止出错。核心 JavaScript 的知识还有很多，如果掌握得不牢靠，那么进行 AJAX 开发时就很容易捉襟见肘了。

### 12.2.3 AJAX 的开发利器——Dojo

“蔡佳娃，记得一开始我们提到的 DWR 吗？”

“嗯，是一个开源的类库，主要用来简化业务功能的开发。”

“好，现在我们来谈另外一个开源的 JavaScript 标准库，Dojo。”

作为一个 JavaScript 的类库，Dojo Toolkit 提供了很多成熟且实用的 JavaScript 部件，开发人员可以随时引入使用。在 AJAX 的开发中，与服务器通信主要是使用 XMLHttpRequest 对象，Dojo 的最大特点之一就是向开发人员隐藏了对 XMLHttpRequest 对象的操作，使得开发人员只需要决定何时通信即可，不需要关注通信的细节。

针对长久以来困扰 JavaScript 开发人员的不同浏览器兼容性差异的问题，Dojo 也通过后台的一些手段将其解决，所以说 Dojo 是一个比较全面的 JavaScript 开发框架。

“师兄，你许诺的 AJAX 的绚丽界面在哪里啊？这么半天讲来讲去总是理论，我可是还没感受到酷炫的效果呢。”

“呵呵，别着急，Dojo 马上给你奉上一些实用美观的组件来。”

“好，我拭目以待！”

前台显示方面，Dojo 通过为多种 Widget 组件提供实现，使得借助于 Dojo 可以很轻松地开发出具有 Web 2.0 特性的华丽页面，最典型的组件如论坛博客系统中必不可少的编辑器控件、搜索网站中的自动搜索匹配输入框 auto-complete combobox 等，图 12-31 列出了一些常用组件的视觉样式。

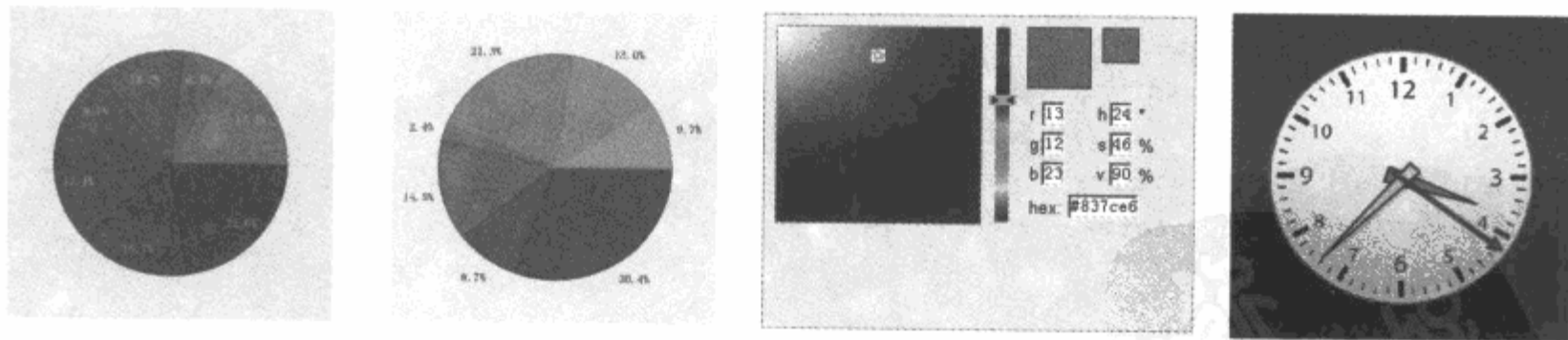


图 12-31 一些 Widget 组件：统计饼图、调色板、桌面时钟

“师兄，我比较感兴趣的还是那个鼠标放上去图标变大的 FishEye 效果，我觉得那个才是比较拉风的呢。”

“OK，今天不仅让你看到 FishEye 是怎样的，还让你学会如何开发自己的 FishEye 效果，让你以后想拉风的时候不用求人也可以自己大摆 pose 了。”

下面将以蔡佳娃感兴趣的 FishEye 为例介绍使用 Dojo 开发 AJAX 应用的一般过程。进行开发之前首先要从 Dojo 的官方网站 [www.dojotoolkit.org](http://www.dojotoolkit.org) 下载工具包，目前最新的正式版本为 1.3.2，其

对应的压缩包为 dojo-release-1.3.2.tar.gz。

下载完成后，先将其解压缩到具体 Web 应用的根目录下，如本例中的 Web 应用叫做 FishEye\_App，然后组织 Web 应用的目录结构，如图 12-32 所示。

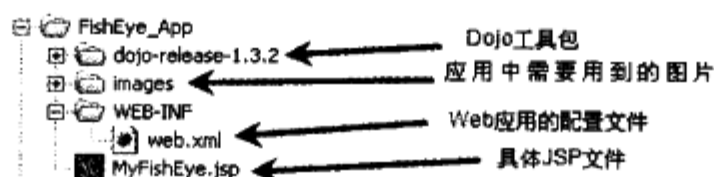


图 12-32 FishEye\_App 目录结构

组织好应用目录之后，就可以开始代码的编写了。打开 MyFishEye.jsp 文件，首先在其中输入以下代码。

```

1  <%@ page contentType="text/html; charset=GBK" %>
2  <html>
3      <head>
4          <title>FishEyes——水平</title>
5          <style type="text/css">
6              //未来添加相关 css 代码
7          </style>
8          <script
9              type="text/javascript"
10             src="dojo-release-1.3.2/dojo/dojo.js"           //引入 Dojo 工具包
11             djConfig="isDebug:false, parseOnLoad: true"> //配置工作模式参数
12          </script>
13          <script type="text/javascript">
14              dojo.require("dojox.widget.FisheyeList");      //指定需要使用的 Widget
15              dojo.require("dojo.parser");                    //启动解析
16          </script>
17      </head>
18      <body class="tundra">
19          <br><br><br><br>
20          <center>
21              <div class="outerbar">
22                  //未来添加 FishEye 对应的 div 代码
23              </div>
24          </center>
25      </body>
26  </html>
  
```

- 上述代码搭建出了整个 Web 页面的框架，其中在第 5~7 行对页面的 CSS 样式进行了定义，这段省略的代码将会在后面提到。
- 第 8~11 行是对 Dojo 工具包中 JavaScript 类库的引入。
- 第 12~15 行指定了需要用到的 Widget 组件，第 20~22 行省略的用于添加 FishEye 组件的具体代码会在后面给出。

AJAX 之所以能向用户展现出绚丽的界面，主要依靠的还是 CSS，本例中的 CSS 样式定义如下。

```

1  @import "dojo-release-1.3.2/dojo/resources/dojo.css";/*Dojo 工具包中 CSS 的引入*/
2  @import "dojo-release-1.3.2/dijit/themes/tundra/tundra.css";
                                   /*Dojo 工具包中 CSS 的引入*/
3  @import "dojo-release-1.3.2/dojox/widget/FisheyeList/FisheyeList.css";
                                   /*Dojo 工具包中 CSS 的引入*/
4  .dojoxFisheyeListBar {          /*FisheyeBar 的外观*/
5      margin: 0 auto;
6      text-align: center;        /*文字居中*/
7  }
8  .outerbar {
9      background-color: #fff;    /*设置背景色*/
10     text-align: center;        /*文字居中*/
11     position: relative;        /*相对定位方式*/
12     left: 0px;
13     top: 0px;
14     width: 100%;               /*大小尺寸*/
15 }
16 body {                          /*设置 body 风格*/
17     font-family: Arial, Helvetica, sans-serif; /*设置字体*/
18     padding: 0;
19     margin: 0;
20     background-color:#fff;      /*设置背景色*/
21     background-image:none;
22 }

```

上述代码首先将 Dojo 工具包中自带的实现 FishEye 效果的 CSS 样式文件引入进来,然后自定义了几种样式,将其应用到 FishEye 组件和正文等地方。

在本例的页面中, FishEye 的添加代码放在“<div></div>”标记中被显示,具体的代码如下所示。

```

1  <div dojoType="dojox.widget.FisheyeList"    <!--指定组件类型-->
2      itemWidth="80" itemHeight="80"          <!--组件中每个元素大小-->
3      itemMaxWidth="180" itemMaxHeight="180"  <!--组件中每个元素大小上限-->
4      orientation="horizontal"                <!--排列方式-->
5      effectUnits="2"                         <!--被选定元素改变时影响周围元素的个数-->
6      itemPadding="10"                       <!--组件中每个元素之间的间隔-->
7      attachEdge="top"
8      labelEdge="right"
9      id="fisheyel">
10     <div dojoType="dojox.widget.FisheyeListItem"
11         onClick="alert('您选择了浏览器');"  <!--自定义的鼠标单击事件处理函数-->
12         label="浏览器"                      <!--鼠标悬停时的说明文字-->
13         iconSrc="images/icon_browser.png"> <!--图片路径-->
14     </div>
15     <div dojoType="dojox.widget.FisheyeListItem"
16         onClick="alert('您选择了日历');"    <!--自定义的鼠标单击事件处理函数-->
17         label="日历"                        <!--鼠标悬停时的说明文字-->
18         iconSrc="images/icon_calendar.png"> <!--图片路径-->

```



```

19     </div>
20     <div dojoType="dojox.widget.FisheyeListItem"
21         onClick="alert('您选择了电子邮件');">        <!-- 自定义的鼠标单击事件处理函数-->
22         label="电子邮件">                                <!-- 鼠标悬停时的说明文字-->
23         iconSrc="images/icon_email.png">                <!-- 图片路径-->
24     </div>
25     <div dojoType="dojox.widget.FisheyeListItem"
16         onClick="alert('您选择了文本编辑器');">        <!-- 自定义的鼠标单击事件处理函数-->
27         label="文本编辑器">                                <!-- 鼠标悬停时的说明文字-->
28         iconSrc="images/icon_texteditor.png">            <!-- 图片路径-->
29     </div>
30     <div dojoType="dojox.widget.FisheyeListItem"
31         onClick="alert('您选择了在线更新');">            <!-- 自定义的鼠标单击事件处理函数-->
32         label="在线更新">                                <!-- 鼠标悬停时的说明文字-->
33         iconSrc="images/icon_update.png">                <!-- 图片路径-->
34     </div>
35     <div dojoType="dojox.widget.FisheyeListItem"
36         onClick="alert('您选择了用户交互');">            <!-- 自定义的鼠标单击事件处理函数-->
37         label="用户交互">                                <!-- 鼠标悬停时的说明文字-->
38         iconSrc="images/icon_users.png">                <!-- 图片路径-->
39     </div>
40 </div>

```

以上代码首先对 FishEye 组件做一些参数的配置，如每个元素的大小和排列方式等，然后对组件中的每个元素进行配置，包括制定图片路径、设置说明文字、挂接鼠标单击事件处理函数等。由于本例重点在于介绍 FishEye 组件的开发过程，所以鼠标单击处理函数非常简单地调用了 alert 方法。

代码编写完成后，启动 Tomcat，在地址栏中输入 `http://localhost:8080/FishEye_App/MyFishEye.jsp`，运行效果如图 12-33 所示。



图 12-33 FishEye 特效示例

“哈哈，我终于也知道怎么开发一个 FishEye 效果的页面啦！”

“哎，瞧你这点出息。不过 Dojo 是个强大的工具，除了看着很有面子的华丽功能，Dojo 也集成了许多异步传输的技术。这些虽然不中看，但可是非常中用的哦。”

服务器推送技术就是一项十分有用的技术，是 AJAX 的重要组成部分。采用服务器推送技术客户端浏览器无须主动发送请求从服务器端“拉”数据，服务器有新的数据后可以向客户端“推”数据，这很类似于传统的基于 Socket 技术的桌面程序客户端。

服务器推送技术的出现提供了一种新的 Web 应用开发方式，同时也大大提升了用户体验，避免了需要实时获得数据的页面定时频繁刷新页面带来的闪烁，同时也降低了 Web 服务器的无用负载。

Comet 技术是最常用的服务器推送技术，本书在第 7 章曾对其做过简单的介绍，在此不再赘述。Dojo 框架中包含了 Comet 标准库，其路径如图 12-34 所示，在开发时只需要将其引入即可。



图 12-34 Comet 包在 Dojo 工具包中的位置

Comet 技术非常适合于开发需要实时信息交互的 Web 应用，比如生产监控、实时通信、股票期货涨跌行情等。用 Comet 技术开发出的 Web 应用将使用户感受到像是普通桌面程序那样良好的交互性，图 12-35 给出了一个使用 Comet 技术开发的某焦化厂生产监控的示例界面。

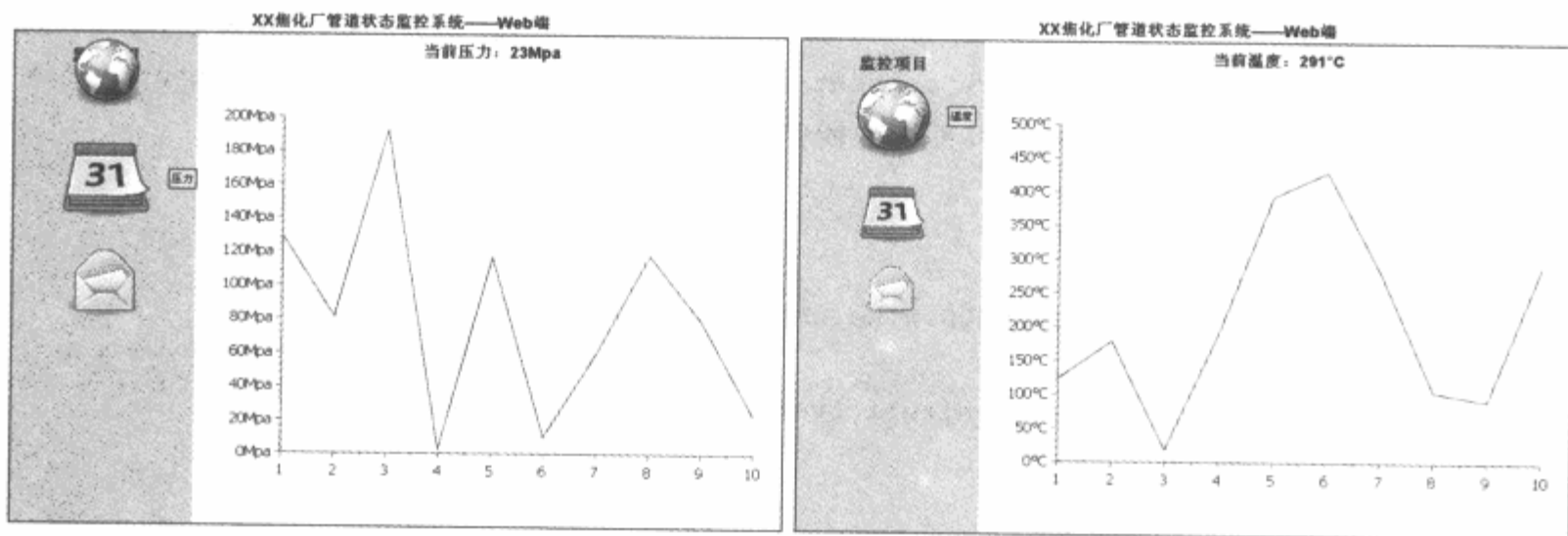


图 12-35 Comet 应用实例——某焦化厂监控系统

**提示** 使用 Comet 进行开发的过程比较复杂，包括服务端和客户端在内有许多工作要做，故本书在此不予详述，有兴趣的读者可以自行到网上搜集资料或查询笔者的相关文章。

#### 12.2.4 AJAX 的最酷代表作——GoogleMap

“哎，说到这我才想起来，其实我们都忘记谈一个很重要的 AJAX 应用了。”

“哦，那是哪个啊？”

“GoogleMap 啊，这应该是提到 AJAX 首先要想到的一个绝佳的例子啊，就像提到巴西第一个念头是足球一样。”

“哦？GoogleMap 在 AJAX 技术发展中有这么重要的地位吗？”

就在 AJAX 技术在 Web 领域初现霸气的时候，很多大的软件厂商如 Google、Sun、Microsoft 等以及一些如 Apache 的开源组织都对其表现出了很大的热情，AJAX 技术因此也得到了非常有力的支持。

在众多争相表现 AJAX 技术优越性的实际 Web 应用中，GoogleMap 算是 AJAX 最得意的代表作品之一。GoogleMap 是由 Google 提供的全球地图信息系统，有传统地图和卫星地图可供选择，

相信几乎所有知道的人都曾经试着在 GoogleMap 上寻找过自己的家乡。

GoogleMap 可贵之处不仅仅是其在诠释 AJAX 技术上的酷炫效果，更重要的是在于其可以支持基于 AJAX 进行二次开发。这就使得 GoogleMap 这个借助 AJAX 赢得天下的应用现在又反过来帮助开发人员去实现 AJAX。下面给出一个使用 AJAX 进行 GoogleMap 简单二次开发的小例子，其过程如下。

① 首先创建一个 HTML 文件，取名为 GooleMapExample.html，需要注意的是在创建页面文件时要将其编码方式设置为“UTF-8”，否则浏览器可能不会正常显示中文。文件编码的设置方式有很多种，最常见的是用 Windows 自带的记事本打开文件，在另存为对话框中选择“UTF-8”并保存。

② 创建好 HTML 文件之后，首先在其中输入以下代码。

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml">
4      <head>                                <!--下面一行为定义字符编码为 UTF-8-->
5          <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
6          <title>Google Maps Example</title>
7          <style>                            <!--定义风格样式-->
8              #mapBao
9              {
10                  width:600px;height:450px;float:left;
11                  position:relative;border:3px solid #65b4f6;
12              }
13          </style>
14          <!-- 未来添加核心代码      -->
15      </head>
16      <body onload="load()" onunload="GUnload()"><!--挂接页面载入、载出事件处理函数-->
17          <div id="mapBao">
18              <div id="map" style="width: 600px; height: 450px"/>
19                  <!--此 div 用于地图显示-->
20          </div>
21      </body>
22  </html>

```

③ 上述代码的作用是将整个界面搭建起来，在上述代码的第 14 行加入如下代码。

```

1  <script src="http://maps.google.com/maps?file=api&v=2&
2      key=ABQIAAAA0DBHEJH01_0VNMudvDaa6xTnXSjFu29e           //API Key
3      Jd6cbMAcBwfVDrkn8RT0twnL4wJ4A6GZWdE82cQ6geLv4Q"       //API Key
4      type="text/javascript"></script>
5  <script type="text/javascript">
6      function load() {
7          if (GBrowserIsCompatible()) {
8              var map = new GMap2(document.getElementById("map")); //创建地图对象
9              map.setCenter(new GLatLng(39.92, 116.46), 12);
10                  //设置地图中心点经纬度及缩放比例

```

```

10      map.addControl(new GSmallMapControl());      //添加小地图控制条
11      map.addControl(new GOverviewMapControl());    //添加缩略图控制
12      map.addControl(new GScaleControl());          //添加比例尺
13      map.addControl(new GMapTypeControl());
           //添加地图类型（卫星、地图、混合）控制
14      }}
15  </script>

```

**提示** 请读者特别注意的是,上述代码第 2 行和第 3 行中有下划线的部分是笔者申请的 API Key,读者若有需要可以去 Google 的官方网站申请自己的免费 API Key。

④ 上述工作完成后,就可以在浏览器中打开页面了,整个代码的运行效果如图 12-36 所示。

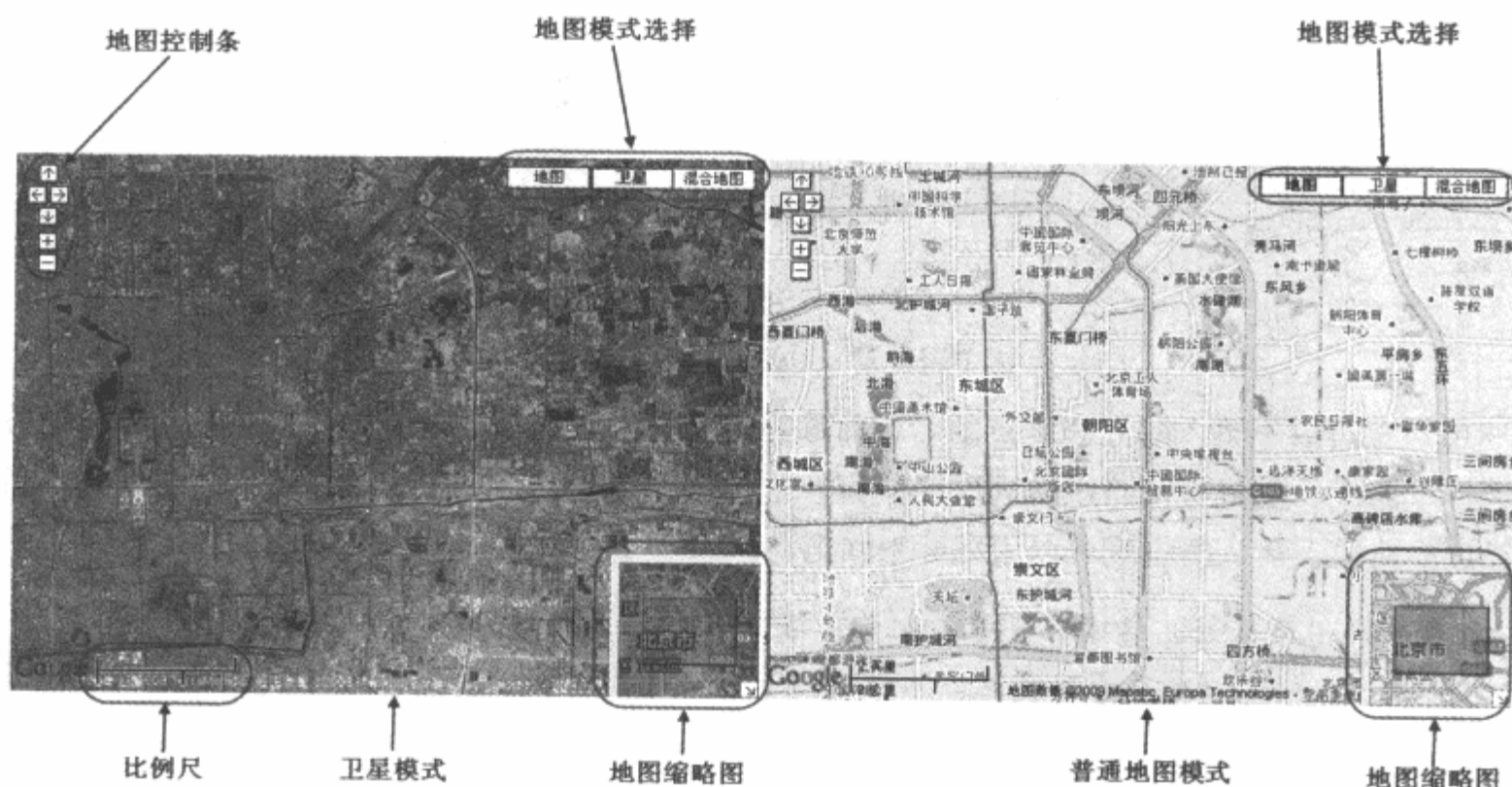


图 12-36 GoogleMapExample 运行效果

该程序通过简单的二次开发,将 GoogleMap 移植到自己的 Web 页面中,而且享受到的功能和直接访问 GoogleMap 的效果是一样的。

“师兄,不是我说你啊,这个程序虽然也很炫、很酷,但也只是简单地调用了一下 GoogleMap 嘛,你也没做什么嘛。”

“呵呵,看来是越来越喂不饱你的胃口了啊,我给你看这个只是因为它简单,而且代码也短,你可以对它有个直观的认识。不过既然你有些不屑,为了保住我身为师兄的尊严,我只好请出一个最近写好的应用给你过过目了,瞧好了啊!”

基于 GoogleMap 如此强大的平台,再充分使用 AJAX 技术,可以开发出更加复杂、用户体验更好的 Web 应用来。如图 12-37 所示的这个行车路线查询应用,就是一个基于 GoogleMap 比较复杂、功能也比较强大的 AJAX 应用,这也是牛开复被蔡佳娃逼到无奈抛出的一个重磅炸弹。

在如图 12-37 所示的应用中,用户可以在屏幕的下方输入起点和终点地名,单击“导航”按钮后,程序将会自动生成起点到终点的行车路线并将其标识到地图上。同时在屏幕的右边栏列出行车路线的详细信息,如距离总长、行车时间估算、所经过的路段等。

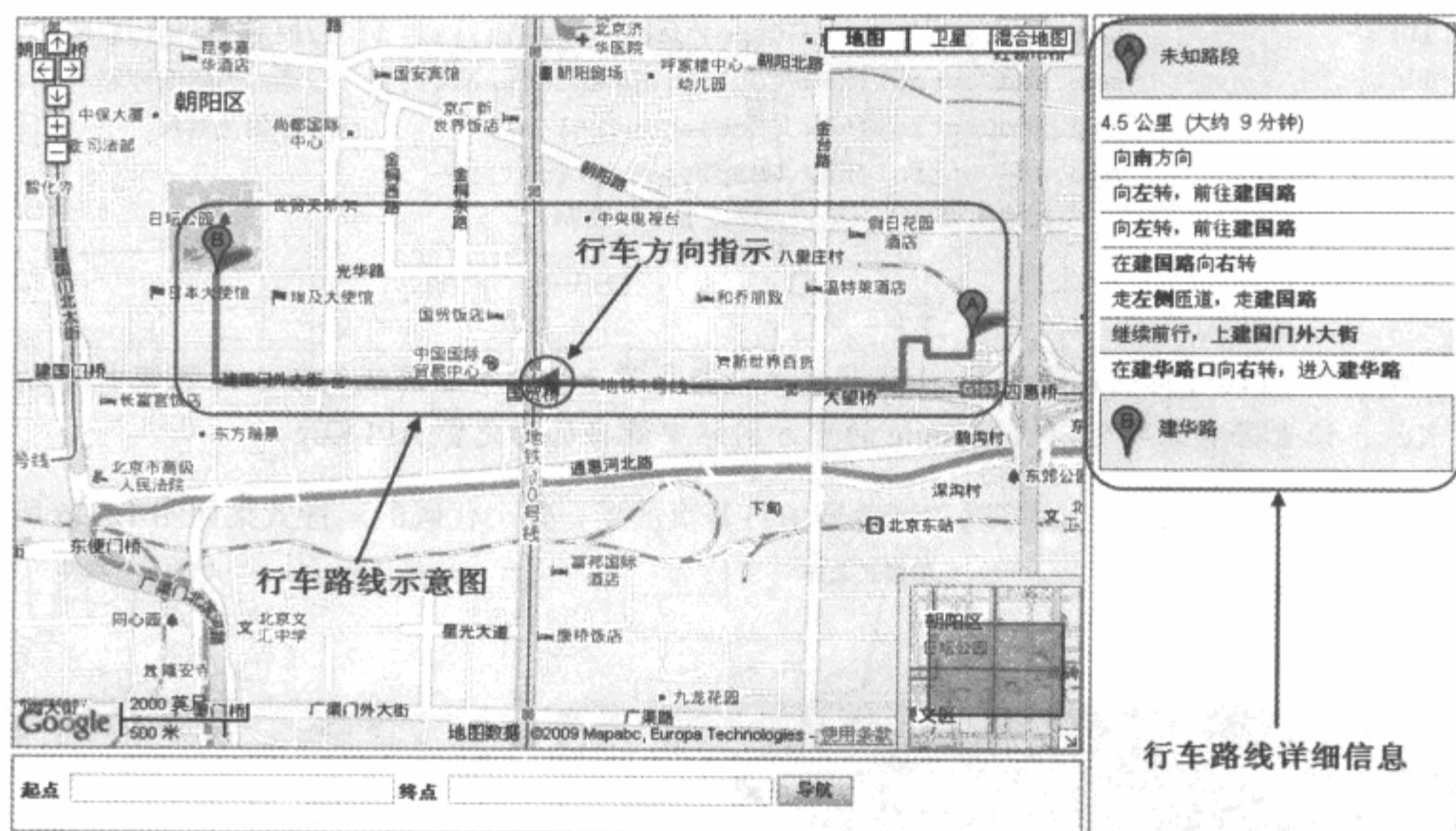


图 12-37 行车路线查询应用

用户也可以不输入起点和终点的名称而选择直接在地图上单击起始、终止位置, 这样也可以生成行车路线, 生成行车路线的同时也会有一个指示方向的箭头沿着路线移动。而右侧将列出正在经过的路段, 箭头的位置可以用鼠标来回拖动。

“怎么样, 蔡佳娃, 这回在下能否把你折服呢?”

“师兄, 我错了, 我愿意用我连绵不绝的泪水来表达对你的崇拜。你做的这个应用简直高妙极了嘛。”

“不止这些哦, 我这两天正考虑将 Google 的 Street View 加进来, 这样在查看行车路线的时候还可以顺便看看路旁的风景呢。”

“师兄, 你说的都已经超出了我的想象力了。”

“呵呵, 所以说 GoogleMap 作为一个 AJAX 技术的成功运用实例和开发平台, 是我们不可多得的一个 AJAX 开发的备用武器库呢。”

### 12.2.5 Web 2.0 时代的异军突起——Mashup

“蔡佳娃, 既然我们提到了 GoogleMap, 那么和它有关的一个技术就不得不提了。”

“哦, 那是哪个技术啊?”

“这个技术就是 Mashup, Mashup 是 Web 2.0 平台下的一个非常好的技术, Mashup 目前国内应用的还不算多, 但在国外已经比较火了。”

“Mashup 是干什么的啊? 学起来容易吗? 和 AJAX 有类似的地方吗? 不会用的是新的编程语言吧?”

“Mashup 可是大大不同于我们之前提到过的 AJAX, 听我介绍完你就明白了。”

Mashup 是一种全新的 Web 应用程序的开发方式, 其主要内容就是将互联网上已存在的 Web



应用、信息和服务根据自己的需要进行自定义的组合和二次开发，形成一个新的 Web 应用程序。“Mashup”原指混合不同风格的音乐以产生新元素，在这里可以理解为“混搭”，Mashup 是 Web 2.0 时代一个重要的发展方向。

与其他技术不同的是，Mashup 应用程序的开发不需要对 Web 应用的具体开发知识有非常深的了解，因为开发一个 Mashup 需要做的工作只是将现有的 Web 应用进行编织，开发者只需要具备一些将各种服务、信息进行整合的能力就可以开发出完全依照自己意愿呈现的酷炫 Web 应用。

“蔡佳娃，Mashup 所体现的是完全自主的思想。你平时上网肯定不会局限于某一个网站吧？”

“当然不是啦，我搜索资料去 Google，查看邮件就去 126，逛逛技术论坛就去 CSDN，看看体育新闻就去新浪，和朋友们联系就来 QQ 空间……哇塞，我每天要浏览的网站还真不少啊。”

“是啊，所以说网络上没有哪个网站是可以为用户提供涵盖所有方面的信息服务的。而如果要根据自己的需求对各种 Web 应用进行取舍和编排，使用 Mashup 就可以了。”

“那 Mashup 的开发到底需要什么技能呢？”

Mashup 的蹿红与 Web 2.0 大旗的迎风飘扬是分不开的，Web 2.0 带来的不仅是 RIA 风暴，还将传统的信息提供改为了信息共享。在传统的 Web 应用中，信息由供应商提供，如网易、搜狐等门户网站。而在 Web 2.0 的应用中，所有的用户既是信息的消费者，又是信息的提供者，如 Youtube、Flickr 以及各种博客都体现了这种 Web 平民化的思想。

在这种趋势下一些网站如 GoogleMap、Yahoo、Flickr 等将自己的部分 Web API 公开给广大开发者，这些 API 非常类似于 SOA 中暴露出的服务点。有了这些 Web API，开发者就可以使用诸如 HTTP 等网络通信协议以及 AJAX 引擎的技术将网站上有用的服务“据为己有”，为我所用。

下面给出一个 Mashup 的开发实例，该 Web 应用通过对 Yahoo、GoogleMap 和 Flickr 提供的不同 Web 应用进行编织，完成了对旅游景点多方位的信息查询，如图 12-38 所示。

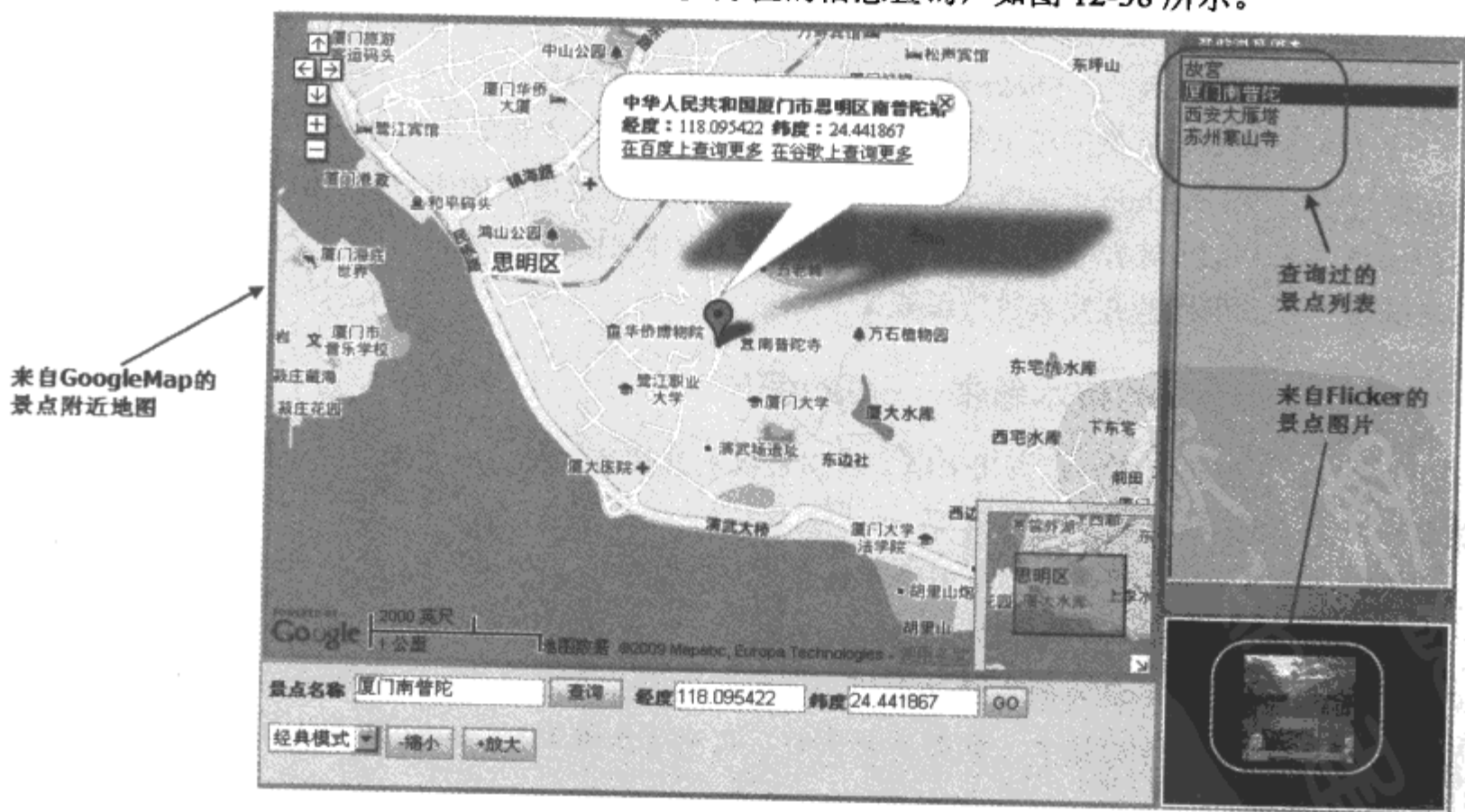


图 12-38 旅游景点多方位信息查询应用

在如图 12-38 所示的 Mashup 应用中, 旅游景点的地图来自于 Google Map, 而屏幕右下方幻灯片显示的图片则来自于 Flickr。输入景点名称后, 可以通过 Yahoo 的定位服务获取该景点的经纬度坐标, 将坐标传给 GoogleMap, 就会得到该景点的地图, 同时将来自于 Flickr 的图片以幻灯片的形式展现。

本例中使用 Google Feed 与 AJAX 的结合, 通过 Internet 从 Flickr 上即时获取对应旅游景点的图片信息。当用户单击图 12-38 中的景点图片时, 自动跳转到 Flickr 的相关图片页面, 如图 12-39 所示。



图 12-39 单击景点图片后跳转到 Flickr 的相关页面

Flickr 是 Internet 上非常著名的一个图片分享网站, 其中的图片资源异常丰富, 几乎囊括了世界各地的各个方面。因此, Flickr 经常被集成到 Mashup 的应用中。

人们对于网络所提供的信息的需求, 往往是全方位的、立体的, 而 Mashup 就可以满足用户的这种需求。使用 Mashup 集成 Web 应用时可以尽情选择质量最好的服务(如地图选择 GoogleMap、图片信息选择 Flickr 等), 这样真正做到了博采众长。同时这些 Mashup 应用又可以放到网上, 作为新的混搭元素供别的开发者进行编织。

### 12.2.6 RIA 殿堂的技术新贵——JavaFX

“蔡佳娃, 关于富客户端应用 RIA, 我们大概也就谈这些吧, 临近结尾了, 按理说我们也应该展望一下未来了。”

“哦, 那师兄你展望展望, 望到啥了给我讲讲啊。”

“呵呵, 我还真望到了一个新生技术, JavaFX。以我个人之见, JavaFX 才是真正酷炫的真谛, 未来的 RIA 必将有 JavaFX 的半壁江山。”

“能得到师兄垂青的技术, 肯定不会差的。我倒是知道 JSF, JavaFX 是什么啊?”

长久以来, Java 凭借其本身的强大在各个领域取得了广泛的应用, 从企业级的 Java EE 到移动平台上的 Java ME, Java 程序运行在全世界许许多多的设备上面。但是在玩酷和打扮方面, Java

却一直退居次席，默默忍受着“Java 无法开发漂亮界面”的论断。

JavaFX 是 Sun 公司推出的脚本语言，它在 2007 年首度露面，并于 2008 年发布了 1.0 版，目前 JavaFX 的最新版本为 1.2。JavaFX 发布的一个重要目标就是在 RIA 这个目前 Web 开发界最热门的领域中功成名就，并彻底甩掉“Java 无法开发炫酷界面”的帽子。

“师兄，既然 JavaFX 是 Sun 在 Java 界面程序开发上打的翻身仗，那 JavaFX 是个怎样的语言呢？”

“首先 JavaFX 是脚本语言，但不像 JavaScript，JavaFX Script 是声明式的脚本语言。很多时候只需要声明描述问题即可，而不用关心实现细节，这有点类似于非过程化语言。”

“哦，那 JavaFX 可真是强大多了呢。”

“简单来说，JavaFX 就是基于 Java 且面向对象的，所有脚本代码最终将会被编译成 Java 类文件，并运行在 JVM 上面。”

JavaFX 的推出改变了之前依靠 Java Swing 和 Java 2D 技术开发 Java 界面程序的传统模式。在开发酷炫效果方面，JavaFX 一点也不逊色于当今市面上的其他技术，使用 JavaFX Script 开发的应用示例效果如图 12-40 和图 12-41 所示。



图 12-40 JavaFX 应用示例 1

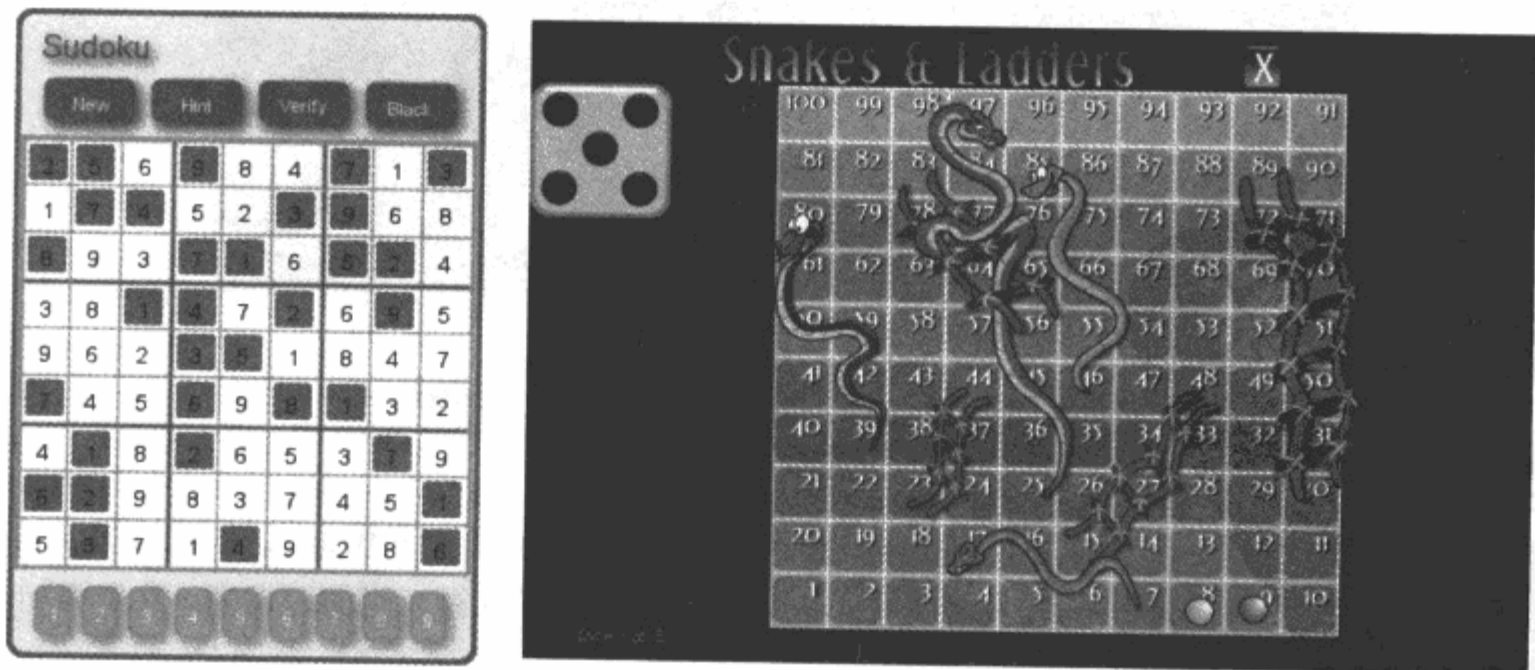


图 12-41 JavaFX 应用示例 2

**提示** 由于本书是黑白灰度印刷，因此读者朋友们看到的图片可能没有那么酷炫。如果真想感受 JavaFX 带来的视觉冲击可以去 JavaFX 官方网站上查看种类繁多的 Demo。

作为 Java 向 RIA 领域发起的蓄谋已久的冲刺，JavaFX 在许多方面都有着不可比拟的优势。

- 由于 JavaFX Script 开发出的应用程序最终将和其他 Java 类一样运行在 JVM 上，因此其得天独厚的跨平台性使得华丽的应用程序不管放到哪个操作系统平台去运行，都能够保证不走形、不变样。
- 在 JavaFX 的脚本中，可以任意调用 Java 类库的 API，Java 的一些诸如网络通信之类的优良特性 JavaFX 都可以将其无一例外地囊括。这势必大大增强了 JavaFX 的后台处理能力，可以说 Java 有多强，JavaFX 就有多强。
- 除了普通 PC，JavaFX 同样也可以运行在移动设备上，伴随着 Android 系统平台的流行，JavaFX 将会是移动设备终端上富客户端程序开发的主流技术。
- 一个华丽的程序界面，除了有后台语言的处理，最基本的图片等视觉元素是必不可少的。JavaFX 深知图形资源对于视觉体验的重要性，提供了一些可供美工设计人员和代码开发人员更好交互的插件。

“师兄，听了你这么一介绍，JavaFX 果然前途无量，可是我斗胆问一下，能不能让俺现场观摩一个 JavaFX 的应用示例呢？”

“早就知道你会这么问了，下面我就用 JavaFX 开发一个简单的立体相册，让你感受一下视觉的冲击。（见图 12-42）”

“太好了，我早就等不及了。”

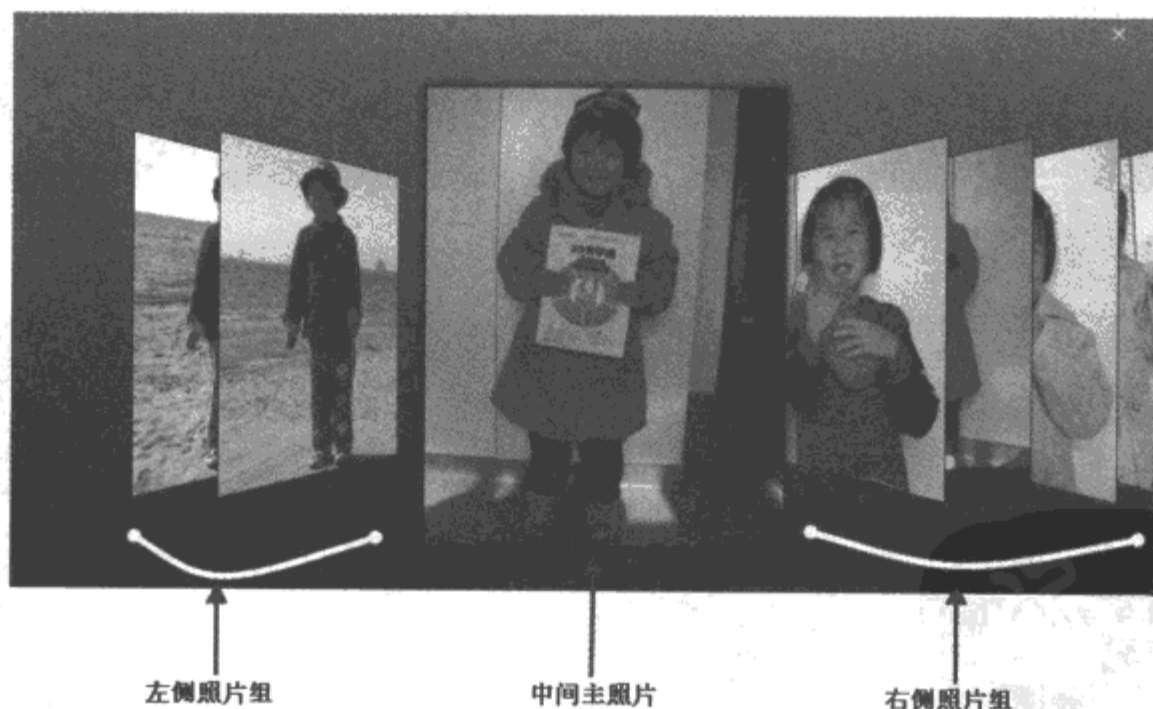


图 12-42 立体相册效果图

进行 JavaFX 脚本开发之前首先需要到官方网站 <http://javafx.com/> 下载 JavaFX 的 SDK，下载完成后进行安装，安装过程与安装 JDK 类似。安装成功后就可以进行应用的开发了，其步骤如下。

### 1. 建立应用目录

为应用程序建立一个名为“MyJavaFX\_3D\_Photos”的目录，本程序的包结构为 wyf.wpf，并

将程序要用到的图片资源所在文件夹 images 和 photos 放到 wpf 目录下。本案例需要建立 3 个 JavaFX 的脚本文件，分别为 Main.fx、Item.fx 和 DisplayShelf.fx，其目录结构如图 12-43 所示。

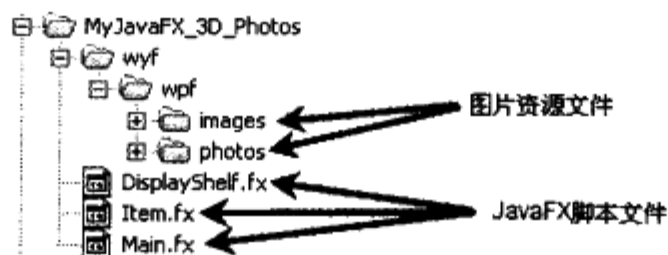


图 12-43 MyJavaFX\_3D\_Photos 应用目录结构图

## 2. 开发 JavaFX 脚本

首先要开发的是 Item.fx 脚本文件，其对应对象在 3D 相册中表示一张一张的照片，也就是说每张照片都是一个 Item 对象。Item.fx 中定义了照片的大小、位置等属性，其代码如下所示。

```

1  package wyf.wpf;
2  import javafx.scene.*;import javafx.scene.image.*;import javafx.scene.effect.*;
                                     //引入相关包
3  import  javafx.scene.input.*;import  javafx.scene.shape.*;import  javafx.scene.
paint.*;                                     //引入相关包
4  import java.lang.Math;                                     //引入相关包
5  public class Item extends CustomNode {                     //继承自定义节点类
6      public var position:Integer = 0;                       //声明位置变量
7      public var angle = 45.0;                               //声明旋转角度
8      public var shelf:DisplayShelf;
9      public-init var image:Image;
10     def width = 240;                                         //照片宽度
11     def height = 320;                                       //照片高度
12     def radius = width/2;                                    //旋转半径
13     def back = height/10;
14     var lx = bind radius - Math.sin(Math.toRadians(angle))*radius - 1;
                                     //值绑定左侧 x 坐标
15     var rx = bind radius + Math.sin(Math.toRadians(angle))*radius + 1;
                                     //值绑定右侧 x 坐标
16     var uly = bind 0 - Math.cos(Math.toRadians(angle))*back;
                                     //值绑定左侧 y 坐标
17     var ury = bind 0 + Math.cos(Math.toRadians(angle))*back;
                                     //值绑定右侧 y 坐标
18     function getPT(t:Number):PerspectiveTransform {
19         return PerspectiveTransform {                       //计算并返回视角
20             ulx: lx    uly: uly
21             urx: rx    ury: ury
22             lrx: rx    lry: height + uly
23             llx: lx    lly: height + ury
24         }}
25     override public function create():Node {               //重写 create 方法
26         return Group {
27             content: [

```



```

28         Group {
29             content: [
30                 ImageView {                                //要显示的图片
31                     image: image
32                 },
33                 Rectangle {                                //图片对应的矩形区域
34                     width: image.width                    //矩形区域宽度与图片宽度相同
35                     height: image.height                 //矩形区域高度与图片宽度相同
36                     fill: Color.TRANSPARENT
37                     stroke: Color.BLACK
38                     smooth: true
39                 }
40             ]
41             effect: bind PerspectiveTransform { //显示效果绑定到视角
42                 ulx: lx    uly: uly
43                 urx: rx    ury: ury
44                 lrx: rx    lry: height + uly
45                 llx: lx    lly: height + ury
46             }},
47             Rectangle {                                //包裹照片的矩形区域
48                 translateX: bind lx                      //x 变换绑定到 lx
49                 width: bind rx-lx                        //宽度绑定到 rx-lx
50                 height: height
51                 fill: Color.TRANSPARENT
52                 blocksMouse: true
53                 onMousePressed: function(e:MouseEvent) {
54                     shelf.shiftToCenter(this);           //鼠标单击后切换此图片为主图片
55                 };
56             }
57         ]
58     }

```

- 上述代码中定义了相册中每幅照片的尺寸、立体效果时旋转的角度，以及鼠标单击某幅照片时让其转化为主照片的动作。
- `PerspectiveTransform` 是 JavaFX 提供的一个系统类，用来表示视角变换，有了它的支持，变换视角的操作就大大简化了。

`Item.fx` 开发完成后，就可以开发 `DisplayShelf.fx` 的源代码了。`DisplayShelf` 是 `Item` 的容器，多幅照片就显示在 `DisplayShelf` 中，其代码如下所示。

```

1  package wyf.wpf;
2  import javafx.animation.*;import javafx.scene.*;           //引入相关包
3  import javafx.util.*;import javafx.scene.effect.DropShadow; //引入相关包
4  public class DisplayShelf extends CustomNode {              //继承自定义节点类
5      public var content:Node[];                               //所有照片数组
6      public-init var spacing = 110;                           //间距
7      public-init var leftOffset = -50;                        //左偏移量
8      public-init var rightOffset = 50;                        //右偏移量
9      public-init var perspective = false;

```

```

10     public-init var scaleSmall = 0.5; //尺寸变换系数
11     var left:Group = Group { }; //左侧照片组
12     var center:Group = Group { }; //中间照片组
13     var right:Group = Group { }; //右侧照片组
14     public var centerIndex = 0; //中间照片索引
15     override public function create():Node {
16         var half = content.size()/2-1;
17         left.content = content[0..half]; //初始化左侧照片组
18         center.content = content[half+1]; //初始化中间照片组
19         center.effect = DropShadow {}; //设置中间主照片显示效果
20         right.content = content[half+2..content.size()-1]; //初始化右侧照片组
21         right.content = Sequences.<<reverse>>(right.content) as Node[];
22         centerIndex = half+1; //计算中间照片索引
23         doLayout(); //进行布局
24         return Group {content: [left,right,center]}
25     }
26     public function reparent(newContent:Node[], newParent:Group):Void {
27         for (n in newContent){
28             if (n.parent instanceof Group) {
29                 delete n from (n.parent as Group).content;
30             }
31             newParent.content = newContent;
32         }
33     public function shift(offset:Integer):Void { //左右移动照片
34         if(centerIndex <= 0 and offset > 0 ){return;} //若中间主照片索引为 0 则返回
35         if(centerIndex >= content.size()-1 and offset < 0 ){return;}
36         //若中间主照片索引为 n-1 则返回
37         centerIndex -= offset;
38         reparent(content[0..centerIndex-1], left); //整理左边组中的照片
39         reparent([content[centerIndex]], center); //整理好中间照片
40         reparent(Sequences.<<reverse>>(content[centerIndex+1..content.size()-1])
as Node[], right);
41         doLayout(); //进行布局
42     }
43     override function doLayout() { //进行布局, 摆放好各张照片的方法
44         var startKeyframes:KeyFrame[]; //起始关键帧数组
45         var endKeyframes:KeyFrame[]; //结束关键帧数组
46         var duration = 0.5s; //时延 0.5 秒
47         for(n in content) { //循环计算每张照片的起始关键帧
48             var it = n as Item;
49             insert KeyFrame { time: 0s values: [
50                 n.translateX => n.translateX,
51                 n.scaleX => n.scaleX,
52                 n.scaleY => n.scaleY,
53                 it.angle => it.angle
54             ] } into startKeyframes; //将计算出的关键帧插入起始关键帧数组
55         }
56         for(n in left.content) { //循环计算左边每张照片的结束关键帧

```

```

56         var it = n as Item;
57         var newX = -left.content.size()*spacing + spacing*indexof n + leftOffset;
58         insert KeyFrame { time: duration values: [
59             n.translateX => newX,
60             n.scaleX => scaleSmall,
61             n.scaleY => scaleSmall,
62             it.angle => 45
63         ] } into endKeyframes;           //将计算出的关键帧插入结束关键帧数组
64     }
65     for(n in center.content) {
66         var it = n as Item;
67         insert KeyFrame { time: duration values: [ //计算中间照片的结束关键帧
68             n.translateX => 0,
69             n.scaleX => 1.0,
70             n.scaleY => 1.0,
71             it.angle => 90
72         ] } into endKeyframes;           //将计算出的关键帧插入结束关键帧数组
73     }
74     for(n in right.content) {             //循环计算右边每张照片的结束关键帧
75         var it = n as Item;
76         var newX = right.content.size()*spacing - spacing*indexof n + rightOffset;
77         insert KeyFrame { time: duration values: [
78             n.translateX => newX,
79             n.scaleX => scaleSmall,
80             n.scaleY => scaleSmall,
81             it.angle => 135
82         ] } into endKeyframes;           //将计算出的关键帧插入结束关键帧数组
83     }
84     var anim = Timeline {keyFrames: {startKeyframes, endKeyframes}};
85                                     //用关键帧生成动画时间序列
86     anim.play();                     //播放照片变换位置时的动画
87     public function shiftToCenter(item:Item):Void {
88                                     //照片移动到中间的方法
89         for(n in left.content) {     //循环对左侧照片处理
90             if(n == item) {
91                 var index = indexof n;
92                 var shiftAmount = left.content.size()-index;
93                 shift(shiftAmount);
94                 return;
95             }
96         }
97         for(n in center.content) {   //对中间照片处理
98             if(n == item) { return; }
99         }
100        for(n in right.content) {    //循环对右侧照片处理
101            if(n == item) {

```

```

102         shift(shiftAmount);
103         return;
104     }

```

- 上述代码将所有的照片划分为三个逻辑组，即左侧照片组、中间主照片、右侧照片组，如图 12-42 所示。当用户用鼠标单击选择不同的照片做主照片时，三个组中的内容会发生相应的变化。
- 另外，在主照片切换时有动画的特效，这在其他语言中可能要经过极其复杂的计算才能实现。但 JavaFX 中引入了类似用 3DMax 做动画时的关键帧技术，开发动画效果时只要指定几个关键帧即可。关键帧之间的非关键帧 JavaFX 会自动生成，大大降低了开发的难度与成本。

最后是开发 Main.fx 的代码，其中包括了主窗体以及其他一些辅助对象的声明，代码如下所示。

```

1  package wyf.wpf;
2  import javafx.stage.*;import javafx.scene.*;import javafx.scene.text.*;
                                     //引入相关包
3  import javafx.scene.shape.*;import javafx.scene.paint.*; //引入相关包
4  import javafx.scene.image.*;import javafx.scene.input.*; //引入相关包
5  var stageDragInitialX:Number;
6  var stageDragInitialY:Number;
7  var dragControl:Group = Group {
8      content:[
9          ImageView { x: 730 y: 8 image: Image { url: "{__DIR__}images/close_rollover.png" }
10             },                                     //关闭按钮图片
11             Rectangle { x: 730 y: 8 width: 10 height: 10 fill: Color.TRANSPARENT
12                 onMouseClicked: function(e:MouseEvent):Void { stage.close(); }
13             }]                                     //关闭按钮单击的有效区域
14  };
15  var images = ["bbta.png","bbtb.png","bbtc.png",
16              "bbte.png","bbtd.png","bbtf.png","bbtg.png"]; //要显示的照片名称数组
17  var half = images.size()/2;
18  var shelf:DisplayShelf;
19  shelf = DisplayShelf {
20      spacing: 57                                     //照片间距
21      scaleSmall: 0.7
22      leftOffset: -140                                 //最左侧位置
23      rightOffset: 110                                 //最右侧位置
24      perspective: true
25      focusTraversable: true
26      content: bind for(i in images) {
27          var item:Item = Item {                       //循环加载每一幅照片
28              angle: 45
29              position: indexof i - half
30              image:Image { url: "{__DIR__}photos/{i}" } //到 photos 目录中取照片
31              shelf: bind shelf

```

```

32     };
33     item;
34 }
35 onKeyPressed:function(e:KeyEvent):Void {           //键盘事件监听
36     if(e.code == KeyCode.VK_LEFT) {               //按下左键头键切换主照片
37         shelf.shift(1);
38     }
39     if(e.code == KeyCode.VK_RIGHT) {               //按下右键头键切换主照片
40         shelf.shift(-1);
41     }
42 var width = 760;                                   //总窗体宽度
43 shelf.translateX = width/2 - 210/2;
44 shelf.translateY = 50;
45 var stage:Stage = Stage {                          //舞台，也就是主窗体
46     title: "Display Shelf"                         //标题
47     visible: true                                  //可见性
48     resizable: false                               //不可改变大小
49     style: StageStyle.UNDECORATED                  //显示风格
50     scene: Scene {
51         content: [
52             Rectangle {
53                 width: width
54                 height: 400                         //总窗体高度
55                 fill: LinearGradient {              //声明线性渐变色填充
56                     startX: 0 startY: 0             //从上侧开始填充
57                     endX: 0 endY: 1                 //到下侧结束填充
58                     proportional: true              //按比例进行
59                     stops: [                        //用渐变色填充主窗体背景
60                         Stop { offset: 0.0 color: Color.rgb(150,150,150) },
61                         Stop { offset: 1.0 color: Color.rgb(30,30,30) },
62                     ]
63                 },
64                 shelf,
65                 dragControl,                        //关闭主窗体按钮所在的工具条
66             ]
67         }
68     shelf.requestFocus();                          //获取输入焦点
69     stage;                                         //显示主窗体

```

将上述三项代码使用“javafx”和“javafx”命令分别进行编译和运行，其运行界面如图 12-44 所示。



#### 提示

从前面的案例中读者朋友们可以体会到，用 JavaFX 开发富客户端效果是非常方便的，代码比采用传统的语言如 Java、C++ 要短很多。可以想象，若直接用 Java 来开发上述效果的界面没有上千行代码是很难做到的。若读者有兴趣学习 JavaFX，可以到官方网站上下载文档或购买 Sun 中国技术社区的推荐书籍《RIA 开发权威指南——基于 JavaFX》。



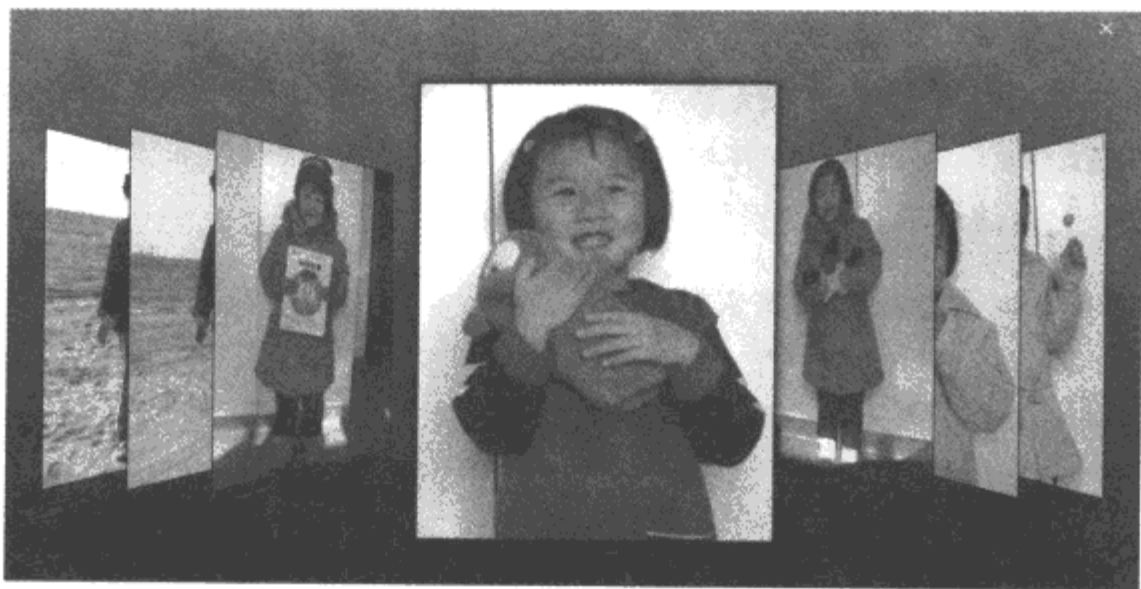


图 12-44 MyJavaFx\_3D\_Photos 最终运行界面

## 12.3 搜索引擎技术

按理说，搜索引擎技术不应该算到新锐兵器谱上，因为搜索引擎从诞生到现在已经将近二十年了，好多互联网巨人就是靠着搜索引擎技术打下江山的，如 Google、百度等。如此“历史悠久”的技术，再拿到本章装嫩也许不会太好使。不过本节将要介绍的搜索引擎技术，并不是如 Google 这般的强大通用搜索引擎，而是基于 Java 技术的一个开源项目 Lucene 以及一个完整的开源搜索引擎解决方案 Nutch。

### 12.3.1 Lucene 开源项目

“师兄，这两天收获真是大啊，先是尝了尝 SOA 这个像法国鹅肝酱般油腻难啃的大餐，又对着 RIA 这个清淡而不失华丽的佳肴‘大快朵颐’了一番，真是太丰盛了。”

“呵呵，不过我还有个甜品没上呢！”

“哦，还有吗？拿出来吧师兄，我今天不饱不归！”

“就看你的肚量行不行了，我们最后来谈谈搜索引擎技术。”

“师兄这哪是甜点啊？搜索引擎都够俺吃一辈子的了！”

“呵呵，我说的可不是 Google 那些搜索引擎啊！身为 Java 程序员，我当然关注的是咱 Java 人自己的搜索引擎项目啦，那就是 Lucene。”

Lucene 是 Apache 软件基金会的一个项目，其 logo 如图 12-45 所示。Lucene 的作者是资深的全文索引/检索专家 Doug Cutting。在详细介绍 Lucene 之前，有必要把当今互联网时代的搜索引擎做一个简单的介绍。



图 12-45 Lucene 的 logo

搜索引擎的发展历经了多个阶段，从 1994 年第一个可以被称作引擎的搜索工具诞生以来，搜索引擎技术的发展经历了人工目录搜索、元搜索、自动搜索等时期。由于搜索引擎技术所涉及的

知识过于广泛，现只将自动搜索引擎做一个浅显的说明。

自动搜索引擎采用的是网络爬虫技术，通俗来讲就是有一种程序每天趴在互联网上，寻找和抓取一些新的 URL 地址和网页快照，并将这些新地址加入到搜索引擎的数据库中，数据库为其建立索引。用户提交查询请求后，搜索引擎后台数据库可以基于索引快速地返回查询结果。

“师兄，我就不明白了，现在的搜索引擎如 Google 和百度已经如此强势了，干嘛还要自己去研究搜索引擎啊，那不是自寻死路吗？”

“非也，人们对于搜索引擎的期待可是不一样的哦。单靠 Google 和百度是养不活全球网民的。”

“哦，师兄那你继续吧。”

当今互联网时代的搜索引擎，大致可以分为如下几种。

- 通用搜索引擎

通用搜索引擎的功能最为强大，因为需要满足用户各种各样的需求，如 Google、百度等都属于通用搜索引擎，这类搜索引擎的用户也是最多的。

- 垂直搜索引擎

通用搜索引擎虽然厉害，但是毕竟面向的客户群太广，很难为某些客户提供深层次的专业服务，众口难调之时，垂直搜索引擎的优势就凸显出来了。垂直搜索引擎关注的是某一个行业，如汽车、招聘、工业原材料等，通过对其进行深入细致的搜索来满足用户在单一方面的高质量需求。

- 内部搜索引擎

往往一个企事业单位的规模变得庞大之后，其内部的文档资料也会浩如烟海。这个时候为了方便对文档资料的检索和处理，就需要在企事业内部设立内部搜索引擎，否则管理起来将会相当困难。

- 特殊搜索引擎

特殊搜索引擎是指专门用来进行特定搜索任务的引擎，比如公安机关需要对网上的违法犯罪信息进行监督和截取，以防止不法分子利用互联网进行犯罪。

以上几种搜索引擎所体现的不同搜索需求，单靠一个通用搜索引擎肯定是无法胜任了。于是自主开发搜索引擎就显得很有必要了，但是搜索引擎是一个门槛非常高的技术，所涉及的内容十分复杂。不过随着 Lucene 开源框架的出现，搜索引擎难于实现的状态逐步得到了改善。

Lucene 是基于全文检索的一个开源项目，它并不是一个完备的搜索引擎，只是提供了一个搜索引擎的框架和必要的 API。开发人员可以基于 Lucene 快速开发出符合特定需求的搜索引擎来。Lucene 搜索框架主要包括两个方面：索引建立和基于索引的检索，这也是 Lucene 框架的工作原理。

Lucene 框架有很多优点，除了由 Java 开发而带来的跨平台性外，Lucene 自身提供的搜索功能几乎满足了开发者的各种需求。同时其近乎无瑕疵的框架结构也使得基于 Lucene 开发自定义搜索引擎变得清晰明了，易于操作。

“蔡佳娃，为了证明 Lucene 框架的优秀，我们今天就利用它来搞一把山寨！”

“啊，师兄，原来你还喜欢山寨啊，我们山寨谁？”

“山寨的目标肯定得是最有名的，那就 Google 吧。”

“师兄你太有才了，好，今天我就跟着你好好山寨 Google 一把！”

### 1. 山寨第一步——下载和配置 Lucene

在浏览器地址栏中输入 <http://lucene.apache.org/> 访问 Lucene 的官方网站下载最新的 Lucene 发布包，目前最新版本为 2.9.1。将 lucene-2.9.1.zip 下载完成后解压缩到一个目录下，如 “C:\lucene-2.9.1”。

下载并解压缩完成后，需要配置 Lucene 的环境变量，在环境变量 CLASSPATH 中添加 “C:\lucene-2.9.1\lucene-core-2.9.1.jar” 和 “C:\lucene-2.9.1\lucene-demos-2.9.1.jar”。

### 2. 山寨第二步——组织应用目录结构

本案例运行在 Tomcat 服务器中，所以需要在 Tomcat 的 webapps 目录下建立如图 12-46 所示的项目目录结构。因为本例只是对 Lucene 的一个简单应用，所以 docs 目录下只是一些会被索引和全文搜索的文本文件。

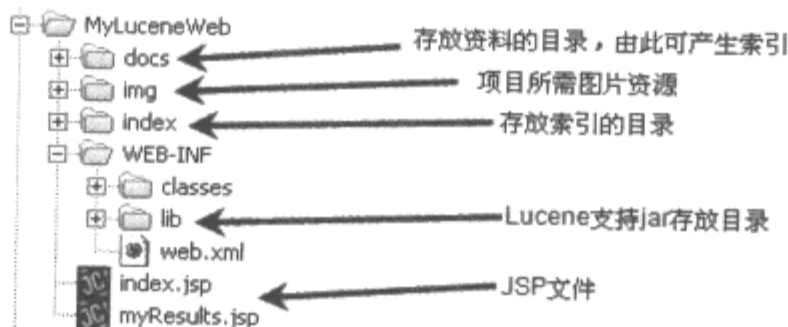


图 12-46 “山寨” Google 的项目目录结构

**提示** 其实目前的各种全文搜索引擎都只能对文本进行全文搜索，所谓能支持多种文件类型不过是增加了将目标类型文件内容提取成文本的功能。

### 3. 山寨第三步——开发后台

开发后台之前需要使用工具通过目标资料文件建立索引，在本例中可以调用 lucene-demos-2.9.1.jar 中的 IndexHTML 类来进行索引的建立。

其操作方式很简单，打开命令行输入以下命令。

```
1 java org.apache.lucene.demo.IndexHTML -create -index 文本文件目录 索引存放目录
```

其中，文本文件目录和索引存放目录分别代表本案例中 MyLuceneWeb 下的 docs 目录路径和 index 目录路径。执行上述命令后，在 index 目录中就会生成基于 docs 目录中资料文件的索引。

索引文件建立完成后，就可以开发基于 Lucene 的后台了。本例中使用一个 Java 类 MyLucene.java 来实现搜索的业务逻辑，其代码如下所示。

```
1 package wyf;
2 import java.util.*;import org.apache.lucene.queryParser.*; //引入相关包
3 import javax.servlet.*;import org.apache.lucene.search.*; //引入相关包
4 import javax.servlet.http.*;import org.apache.lucene.store.*; //引入相关包
5 import java.io.*;import org.apache.lucene.analysis.*; //引入相关包
6 import org.apache.lucene.analysis.standard.StandardAnalyzer; //引入相关包
7 import org.apache.lucene.document.*;import org.apache.lucene.index.*;
//引入相关包
```

```

8   import java.net.URLEncoder;import org.apache.lucene.util.Version; //引入相关包
9   public class MyLucene{
10      static String indexPath = "C:\\Program Files\\apache-tomcat-6.0.14\\"
11          + "webapps\\MyLuceneWeb\\index";
12      static IndexReader iReader;          //负责将索引读入
13      static IndexSearcher iSearcher;      //负责对索引进行检索
14      static{
15          try{
16              iReader=IndexReader.open(FSDirectory.open(newFile(indexPath)),true);
17              iSearcher = new IndexSearcher(iReader);
18              //创建一个 IndexSearcher 成本很高,所以要重用
19          }
20          catch(Exception e){    e.printStackTrace();}
21      public static ArrayList<String []> getSearchResults(String queryString,String
start,String max){
22          ArrayList<String []> al = new ArrayList<String []>();
23          //定义要返回的 ArrayList 对象
24          Query query;
25          TopDocs searchResults = null;
26          int startIndex = Integer.parseInt(start); //结果显示的起始位置
27          int maxResults = Integer.parseInt(max); //最大显示结果个数
28          int pageSize = maxResults;              //此次返回的页面实际显示结果的个数
29          Analyzer myAnalyzer = new StandardAnalyzer(Version.LUCENE_CURRENT);
30          //分析输入关键字
31          try{ //进行查询转换,搜索,将结果保存在 searchResults 里面
32              QueryParser qParser = new QueryParser("contents",myAnalyzer);
33              query = qParser.parse(queryString);
34              searchResults = iSearcher.search(query,maxResults);
35              //生成查询结果集合
36          }
37          catch(Exception e){    e.printStackTrace(); }
38          if(searchResults.totalHits == 0){          //没有搜索到结果
39              String [] sa = {"none","none"};      //设置空信息
40              al.add(sa);
41              return al;                            //返回空信息
42          }
43          if(startIndex+maxResults > searchResults.totalHits){
44              //未显示的结果数小于页面最大显示结果数目
45              pageSize = searchResults.totalHits - startIndex;
46              for(int i=startIndex;i<startIndex+pageSize;i++){
47                  //将结果包装进 ArrayList
48                  String [] tsa = new String[3];
49                  Document doc = null;
50                  try{
51                      doc = iSearcher.doc(searchResults.scoreDocs[i].doc);
52                      //获取文件
53                  }
54                  catch(Exception e){    e.printStackTrace(); }
55                  String docTitle = doc.get("title");//获取文件标题

```

```

49         String docPath = doc.get("path");    //获取文件路径
50         if(docTitle.equals("") || docTitle == null){
                                                    //标题为空，则标题为路径名
51             docTitle = docPath;
52         }
53         tsa[0] = docTitle;                        //文件标题
54         tsa[1] = docPath;                        //文件路径
55         tsa[2] = doc.get("summary");            //文件摘要
56         al.add(tsa);
57     }}
58     return al;
59 }

```

上述代码通过对 Lucene 类库中 API 的一些调用，实现了针对某个查询字符串的搜索工作。这只是对于 Lucene 框架的一个基本操作，很多功能并没有实现，如只能对文本文件而不能对包含文本信息的 HTML 等文件进行检索。

#### 4. 山寨第四步——形似

后台开发完毕后，接下来就是前台的简单开发了，本案例中需要用到两个 JSP，它在项目中的位置可参见图 12-46，其中 index.jsp 的代码如下所示。

```

1  <%@ page contentType="text/html;charset=GBK"%>
2  <html>
3      <head><title>Lucene 搜索引擎示例</title>    </head>
4      <body>
5          <br><br>
6          <center></center>
7          <form action="myResults.jsp" method="post">
8              <p align="center"><input type="text" name="queryString" value="">
9                  <input type="hidden" name="maxResults" value=
"50">
10                 <input type="submit" value="搜索"></p>
11          </form>
12      </body>
13  </html>

```

index.jsp 主要负责显示主页并接受用户输入检索关键字，这也是此次山寨的外观核心环节。另外一个 JSP 文件 myResults.jsp 的代码如下所示。

```

1  <%@ page contentType="text/html;charset=GBK" import="java.util.*,wyf.MyLucene" %>
2  <html>
3      <head><title>搜索结果</title></head>
4      <body>
5          <%
6              String queryString = request.getParameter("queryString");
                                                    //获取搜索参数
7              String maxResults = request.getParameter("maxResults");
8              if(queryString != null){                //若搜索参数不为空则进行搜索
9                  ArrayList<String []> al = MyLucene.getSearchResults(queryString,
"0",maxResults);
10                 String [] head = (String [])al.get(0);

```



```

11             if(head.length == 2){
12                 %>
13                 <hr><center>对不起，没有找到您要的结果</center>
14                 <%          }
15                 else{
16                 %>
17                 <table>
18                     <tr><th>文件</th><th>摘要</th></tr>
19                 <%
20                     for(int i=0;i<al.size();i++){          //循环打印输出搜索结果
21                         String [] tsa = (String [])al.get(i);
22                 %>
23                     <tr>
24                         <td><a href="<%= tsa[1]%>"><%= tsa[0] %></a></td>
25                         <td><%= tsa[2] %></td>
26                     </tr>
27                 <%          }      %>
28                 </table>
29                 <%          }}      %>
30         </body>
31     </html>

```

myResults.jsp 负责接收来自主页的搜索请求，并通过调用 MyLucene 类业务方法 getSearchResults 查询结果并返回给用户。程序运行后的“山寨”主页如图 12-47 所示，在搜索框中输入 java，搜索结果如图 12-48 所示，搜索结果页面中显示的文本文件内容都包含“java”关键字。

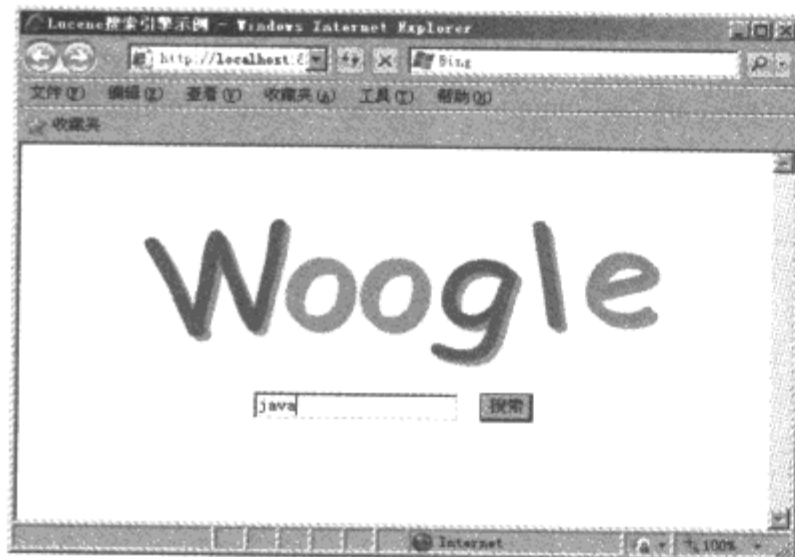


图 12-47 Woogle 搜索主页

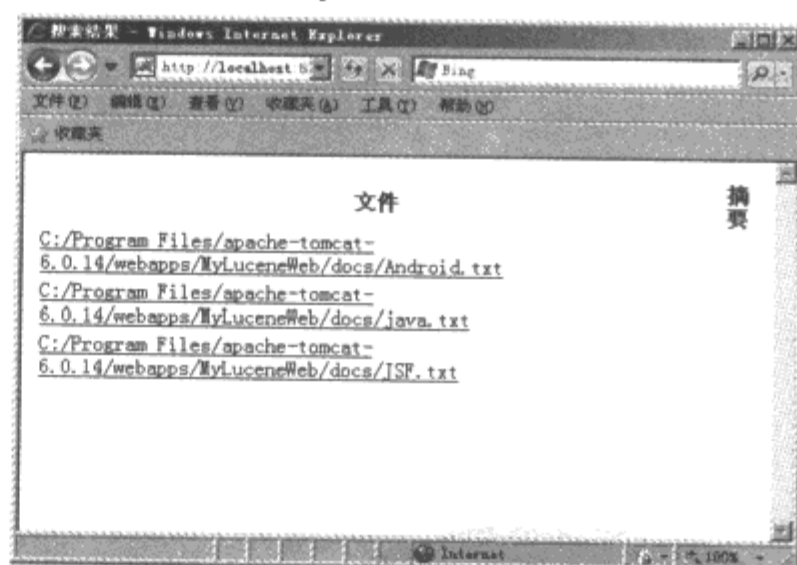


图 12-48 Woogle 搜索结果页面

从这个小例子的开发过程读者朋友们应该可以感觉到使用 Lucene 开源框架来开发自己的搜索引擎是十分方便的，有兴趣从事这方面工作的读者可以进一步研究。

### 12.3.2 Nutch 框架

“介绍完 Lucene 之后，我们来谈谈 Nutch。”

“师兄，Nutch 是什么啊？是另外一种搜索引擎框架吗？”

“非也，Nutch 和 Lucene 有着千丝万缕的联系，且听我慢慢道来。”

Nutch 的 logo 如图 12-49 所示,它是一个用 Java 开发、开放源代码的搜索引擎解决方案。Nutch 的内部是基于 Lucene 的,但 Lucene 只是提供了一些编程接口和框架结构,是个需要开发的半成品,而 Nutch 则是一个完备的搜索引擎系统成品。



图 12-49 Nutch 的 logo

秉承了 Lucene 的优良结构和强大的全文检索功能, Nutch 在易用性方面做得也非常出色,真正让搜索引擎技术可以草根化。同时 Nutch 还有很多优于 Lucene 的功能,如技术上实现了非常复杂的分布式检索等。

由于 Nutch 的开发所涉及的知识很多,本书无法一一详尽介绍,下面主要介绍一下 Nutch 搜索引擎在 Windows 系统上的简单部署过程。

### 1. 下载 Cygwin

由于 Nutch 诞生在 Linux 平台下,所以要想在 Windows 环境中运行,需要有 Cygwin 的支持,否则不能很好地调用 Nutch 的功能。在浏览器地址栏中输入 <http://www.cygwin.com/> 访问 Cygwin 官方网站,并下载最新版本 1.5.25-15 的安装文件。

双击下载到计算机的 setup.exe,会出现提示对话框让用户对安装路径或其他选项做一些配置,然后选择一个下载站点,就开始从互联网上下载并安装 Cygwin。

下载完成后需要对 Cygwin 进行一些测试,启动 Cygwin,在出现的命令窗口中输入一些 Linux 系统命令如 whoami、ls 等,如果 Cygwin 的反应正常,则表示安装成功。

### 2. 下载 Nutch

在浏览器地址栏中输入 <http://lucene.apache.org/nutch/> 访问 Nutch 的官方网站,选择要下载的版本后进入 Nutch 的下载页面,目前最新版本为 1.0。下载完成后直接解压缩到一个目录即可,如“C:\nutch-1.0”。

下载 Nutch 完成后需要对整个运行环境进行测试,启动 Cygwin,并在其命令窗口中输入“cd /cygdrive/c/nutch-1.0”命令,将当前目录切换到 Nutch 的安装目录。其中“/cygdrive/c”是虚拟目录,表示系统的本地磁盘驱动器 C。

切换工作目录后,可以像测试 JDK 是否成功安装一样,执行“bin/nutch”命令。如果出现一些说明命令用法的帮助信息,则表示整个运行环境搭建成功,如图 12-50 所示。

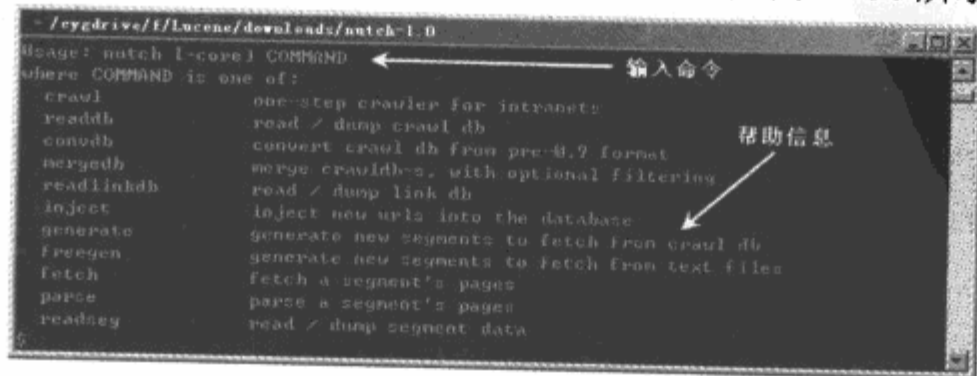


图 12-50 nutch 命令正确执行后输出的提示信息

### 3. 建立索引数据库

要想使搜索引擎工作，必须要有后台的索引数据库，索引数据库是通过前面提到的网络爬虫技术建立的，其所需要的技术与过程比较复杂，而且建立的时间也相对长，本书不再赘述，有兴趣的读者可以参考相关资料自行完成。本例中直接对建立好的索引数据库进行操作，将其放到磁盘的一个目录下，比如“C:\apachewyf”。

### 4. 部署 Nutch

将 Nutch 安装目录下的 nutch-1.0.war 文件复制到 Tomcat 的 webapps 目录下，启动 Tomcat，Tomcat 会将 nutch-1.0.war 自动解压成目录 nutch-1.0。停止 Tomcat，打开 webapps 目录，定位到 nutch-1.0\WEB-INF\classes 目录下，对该目录下的 nutch-site.xml 进行如下编辑。

```

1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3  <configuration>
4      <property>
5          <name>fs.default.name</name>    <!--设置索引数据库用文件方式提供-->
6          <value>file:///</value>
7      </property>
8      <property>                            <!--配置索引数据库-->
9          <name>searcher.dir</name>
10         <value>C:/apachewyf</value>      <!--指定索引数据库位置-->
11     </property>
12 </configuration>

```

配置 nutch-site.xml 的目的是让 Nutch 搜索引擎和索引数据库进行关联。配置完成之后，将目录定位到 Tomcat 根目录下的 conf 文件夹下，打开 server.xml，将“<Connector port="8080" .....>”标记修改为如下代码。

```

1  <Connector port="8080" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
2          enableLookups="false" redirectPort="8443" acceptCount="100"
3          connectionTimeout="20000" disableUploadTimeout="true"
4          URIEncoding="UTF-8" useBodyEncodingForURI="true" /> <!--指定编码方式-->

```

该配置主要是为了使搜索引擎能够支持中文或其他英文以外的字符。最后打开 Cygwin，将目录定位到 Tomcat 的 bin 目录下，使用 startup.sh 命令在 Linux 模式下启动 Tomcat。不建议直接在 Windows 环境中启动 Tomcat，那样有可能会带来一些问题。

启动 Tomcat 之后，在浏览器地址栏中输入 <http://localhost:8080/nutch-1.0>，如果配置正确的话，就会出现如图 12-51 所示的搜索主页面。

在搜索主页面中输入关键字，如 java，并按下“搜索”按钮，就会显示出如图 12-52 所示的搜索结果，这些搜索结果都来自于事先用网络爬虫建立的索引数据库。

“师兄，看来咱们 Java 界的新技术还真是层出不穷啊！”

“是啊，我还怕你没吃饱呢。”

“饱了饱了，都快撑得扶墙走了，这些新技术随便挑出个来研究研究，都得花不少精力呢。”

“呵呵，不过身为我辈中人，对于新技术的渴望，就应该如饥似渴一些嘛。”

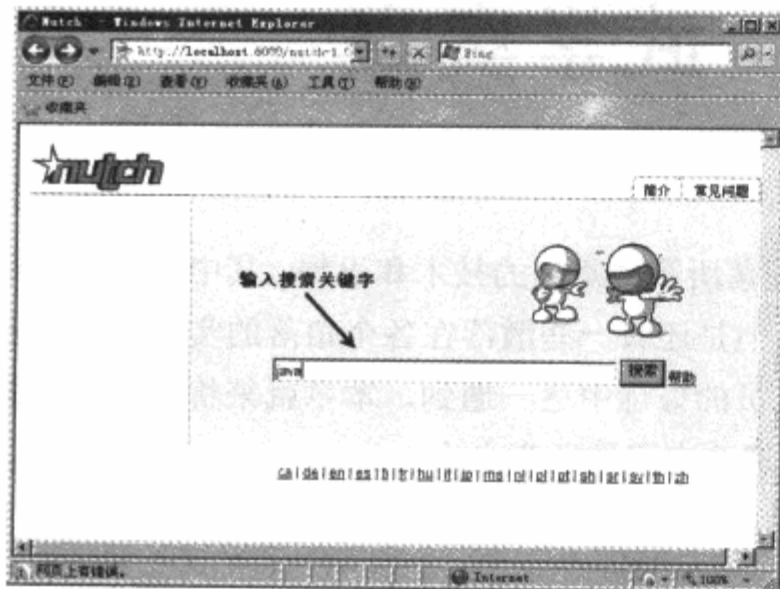


图 12-51 Nutch 搜索主页

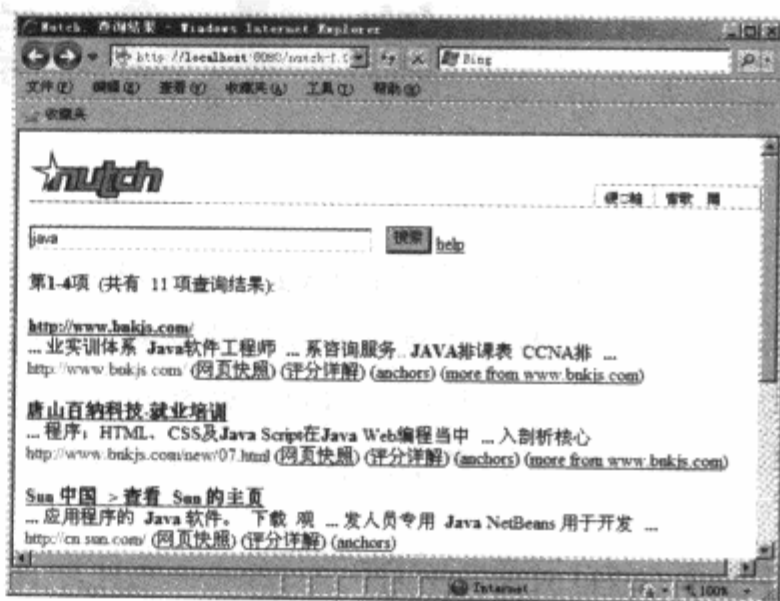


图 12-52 Nutch 搜索结果页面

## 12.4 本章小结

本章是立足当下 Java 世界放眼远观的一个章节，为了让读者对于市面上初露锋芒的新技术有一个全面的了解，本章提供了大量的图片和示例加以描述。这些新技术中有如 SOA 般高深莫测的思想，也有如 RIA 般时尚的前台表示，还有正在平民化的搜索引擎技术。

相信读者朋友也可以通过本章享受到和蔡佳娃一样的技术大餐。由于篇幅有限，本章对于新技术的说明都是简单地介绍思想，所举的例子也非常浅显，有兴趣的读者可以参考相关书籍资料自行研究。

# 第 13 章 武学奥义

前面几章介绍了很多 Java 开发人员在江湖中闯荡所需要掌握的技术和思想，其中既有能一招制敌的实用兵器，也有需要厚积薄发的深厚内功。但是还有一些散落在各个角落的实用兵器和内功心法未曾涉及，这些兵器和心法都将会开发人员的修炼中逐一遇到。本章就来将这些知识做一个简单的介绍，希望读者朋友们可以将其融会贯通到自己的所学之中。

## 13.1 单元测试的利器——JUnit

软件项目的开发中总是伴随着测试，这个环节虽然不够激动人心，但却是必不可少的，否则便无法保证程序在业务逻辑上的正确性和性能上的要求。身为 Java 的开发人员，自然也少不了与测试工具打交道，本节就介绍一款简单而又强大的单元测试工具——JUnit。

### 13.1.1 JUnit 简介

“师兄啊，我今天被经理骂了，说我写代码不注意测试，结果程序运行出错后顺藤摸瓜地找了半天才找到 bug 的位置，哎，惭愧惭愧啊！”

“呵呵，很多像你这样的高级菜鸟都会犯这个错误，认为只要解决了技术上的可行性问题，编程工作仅仅是一个毋庸置疑不需要验证的过程，从而总是不注意去测试所写代码。不培养自己随时测试的开发习惯，这样可是很难成为高人的啊。”

软件测试是软件开发过程中的重头戏之一，不经过测试的软件是没办法交付客户使用的。软件测试穿插在整个项目的生命周期内，在有些编程方法学如极限编程中，测试部分的代码甚至需要在正式代码开发之前就完成。

软件测试从不同规模来划分，有单元测试、集成测试等。其中最需要开发人员关注的就是单元测试，单元测试是在编写代码的过程中进行的，是对所开发代码是否正确和能否满足业务需求的初步检验。因此，可以将单元测试看做是在程序经过编译这道关卡之后的第一个质量监督环节。

单元测试应该作为开发人员编程工作的一部分来对待，执行完备的单元测试可以让后续的开发工作更加驾轻就熟，每个开发人员都有义务对自己的代码进行详细周全的单元测试。

“师兄，单元测试的重要性我是明白了，不过单元测试怎么进行比较好呢？”

“单元测试的方式可以是多种多样的，比如可以直接在某个类中加上个 main 方法然后将测试代码写进去运行，这是最简单的。”

“嗯，这个我也经常用，其他的呢？”

“还有就是用 Java 的断言机制，不过最好的方式还是选用工具进行测试，在 Java 开发中 JUnit 已经近似被看做是单元测试的标准工具了。”

将测试代码写在正式代码中（如在 main 方法中）进行测试，虽然比较直接，但是测试模块过




于分散不容易管理，而且程序交付使用时处于多方面考虑这些测试代码还是需要屏蔽的，所以很不方便。Java 中的断言机制比较好，但是断言的测试能力实在太有限，而且断言很容易被滥用，并造成一些程序上的 bug。

JUnit 可以很好地实现 Java 开发中的单元测试，它是开源的单元测试框架。JUnit 的特点就是小巧，但是功能却很强大，不仅可以对普通 Java 类进行测试，面对 JSP、EJB 等不同的测试对象 JUnit 也可以通过采取相应的规则方便地进行测试。

JUnit 主要是基于断言机制进行测试的，通过对测试结果和期望结果进行比较，验证某个业务方法是否正确。JUnit 框架提供了很多测试方法，灵活地使用它们可以在很大程度上简化测试程序的开发，这些方法的说明如表 13-1 所示。

表 13-1 JUnit提供的静态Assert方法

方    法	说    明
assertEquals	该方法被重载，可以验证两个 float 或 double 等 Java 基本数值类型是否在指定的误差范围内相等，还可以对两个对象或对象数组进行相等性判断
assertTrue	测试某个表达式是否为真，否则测试失败
assertFalse	测试某个表达式是否为假，否则测试失败
assertNull	测试传入的对象引用是否为 null，否则测试失败
assertNotNull	测试传入的对象引用是否不为 null，否则测试失败
assertSame	测试传入的两个对象引用是否指向同一个对象，否则测试失败
assertNotSame	测试传入的两个对象引用是否指向不同的对象，否则测试失败

 对于表 13-1 中的每一个方法，都有相应的重载版本可以输出测试信息。如 assertEquals ( Object a,Object b ), 其有重载版本 assertEquals( String message,Object a,Object b ), 其中 message 表示当测试 fail 的时候输出的信息。上述方法的具体用法本书将会在下一小节中举例说明。

13.1.2 单枪匹马，赤膊上阵——JUnit 的单独使用

“师兄，我还真是孤陋寡闻啊，之前我做开发如果真的有必要去写测试代码，都是直接 System.out.println(), 或者顶多在 main 方法里面啰嗦几句，对于你说的 JUnit 还真是不太熟悉啊。”

“呵呵，不要着急，JUnit 虽然强大，但是易用性特别好。有师兄在，我保证你今天晚饭之前彻底会用 JUnit 进行简单的测试。”

在 JUnit 框架中，用来测试业务代码的测试类被称作测试用例（Test Case），下面给出一个使用 JUnit 框架进行测试的简单示例。本例中被测试的 TestMe 类中定义了一些业务方法，而基于 JUnit 框架开发的 MyTestCase 类和 MyTestSuite 类负责对其进行测试，整体流程如下。

1. 准备工作

进行开发和测试之前，首先要有 JUnit 的 jar 包，可以去官方网站 <http://www.junit.org/> 上下载 JUnit 的工具包 junit4.7.zip（本例以目前的最新版本 JUnit 4.7 为例）。下载完成后将 junit4.7.zip 解压缩，并将其中的 junit-4.7.jar 文件放到指定目录下，如“C:\junit-4.7.jar”。同时，可以在 C 盘根目录下新建 JUnit 目录，用来存放相关的源代码。


## 2. 业务类的开发

首先需要开发的是实现业务功能的类 TestMe，其对应的源文件 TestMe.java 位于“C:\JUnit”中。TestMe 类中定义了接收变长参数并对其进行排序和求平均值的两个业务方法，其代码如下所示。

```

1  package wyf;
2  import java.util.*;                //引入相关包
3  public class TestMe{
4      public int [] varSort(int... varArgs){ //该方法接收变长参数进行排序
5          Arrays.sort(varArgs);           //将参数序列进行排序
6          return varArgs;
7      }
8      public int varAvg(int... varArgs){    //该方法接收变长参数进行平均值计算
9          int avg=0;
10         int length = varArgs.length;     //获取变长参数的个数
11         for(int i=0;i<length;i++){
12             avg+=varArgs[i];             //对参数求和
13         }
14         avg /= length;                   //计算平均值
15         return avg;
16     }}

```

 **提示** 代码第4行和第8行中的“int... varArgs”为两个业务方法各自声明了一个名为 varArgs 的整型变长参数，因此这两个业务方法均可以接收任意个数的整型变量序列作为入口参数。

## 3. 测试用例的开发

业务实现类 TestMe 开发完成后，就可以进行测试用例的开发了。在“C:\junit-4.7.jar”下新建一个名为 MyTestCase.java 的文件，在其中输入如下代码。

```

1  package wyf;
2  import junit.framework.*;import java.util.*; //引入相关
3  public class MyTestCase extends TestCase{ //声明继承自 TestCase 的测试用例
4      public void testSort(){                //对 TestMe 类的 varSort 方法进行测试
5          TestMe tm = new TestMe();
6          int [] expect = {1,2,3,4,5,6,7,8};
7          int [] actual = tm.varSort(4,3,2,6,8,5,7,1);
8          assertTrue(Arrays.equals(expect,actual)); //通过断言判断是否满足测试要求
9      }
10     public void testAvg(){                  //对 TestMe 类的 varAvg 方法进行测试
11         TestMe tm = new TestMe();
12         int actual = tm.varAvg(5,12,9,6);
13         assertEquals("平均值求解错误!",8,actual); //通过断言判断是否满足测试要求
14     }
15     public void setUp(){                    //重写父类的 setUp 方法
16         System.out.println("This is SetUp.");
17     }
18     public void tearDown(){                 //重写父类的 tearDown 方法

```

```
19         System.out.println("This is Tear Down.");
20     }
}
```

- 代码第 4 行和第 10 行分别声明了对 TestMe 类中 varSort 和 varAvg 业务方法进行测试的测试方法。在实际开发中，强烈建议采用如上这种“testxxx”的方法命名规则，同时还要注意将这些测试方法定义成 public void 格式的。
- 代码第 15 行和第 18 行是对父类 TestCase 中对应方法的重写，setUp 方法中定义了进行测试前要做的准备工作，而 tearDown 方法定义了测试结束后需要做的扫尾工作。本测试程序因为比较简单，所以在 setUp 与 tearDown 的方法体中只是简单的打印输出。

对编写完的代码进行编译，没有错误后打开命令行窗口，输入如下命令。

```
1 java -classpath c:\junit-4.7.jar;C:\JUnit junit.textui.TestRunner wyf.MyTestCase
```

上述命令中的“junit.textui.TestRunner”是 junit-4.7.jar 包中的工具类，用来运行测试用例，测试结果的输出如图 13-1 所示。

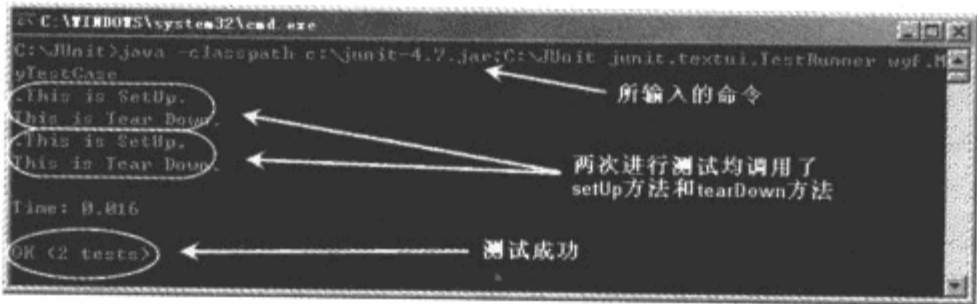


图 13-1 测试结果输出

如果实在厌倦了黑乎乎的命令行，JUnit 也提供了可视化的测试模块，但是需要的版本是 3.8，JUnit 4.0 及其以上的版本中均没有提供这个模块。使用窗口化测试模块进行测试的方式与命令行类似，只是需要输入的测试命令改为如下形式。

```
1 java -classpath c:\junit-3.8.1.jar;C:\JUnit junit.swingui.TestRunner wyf.MyTestCase
```

其中“junit.swingui.TestRunner”表示调用的测试模块为窗口模式，在窗口模式下进行测试的运行结果如图 13-2 所示。

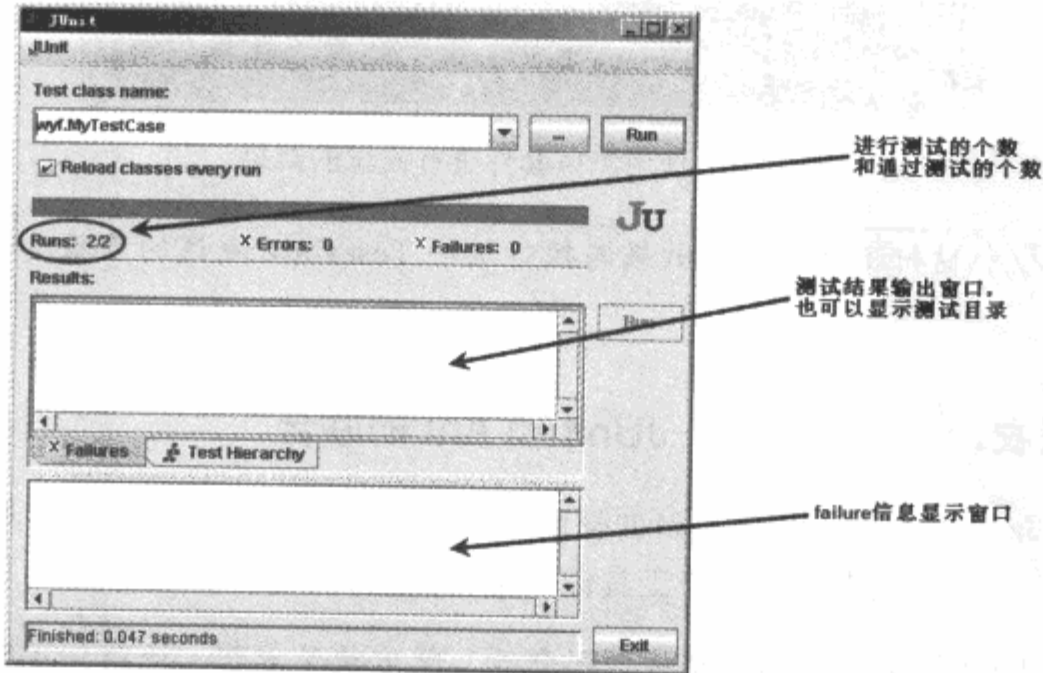


图 13-2 窗口化显示测试结果

需要说明一下在 JUnit 的测试中 Error 和 Failure 的区别。Error 是指在测试中遇到了测试范围之外的错误，如无法找到类文件等；而 Failure 是指测试中所测试代码出现的不符合既定业务逻辑的错误，假设本例中的计算平均值方法测试出错的话，JUnit 就会返回一个 Failure。

“师兄，我再大言不惭一回啊，你这个 JUnit 虽然比较好使，不过一个 TestCase 类中也不能把所有要测试的要素全添上去啊，那样岂不是还要对每个 Test Case 逐一执行？”

“早就想到你会这么问了，在 JUnit 中不仅可以对一个 TestCase 类进行测试，还可以将多个测试用例放到一起来测试，下面我就给你来演示一下。”

#### 4. 在测试套件中进行多项测试

JUnit 中可以基于 TestSuite 开发一个测试套件来执行多个测试用例，本例中测试套件 MyTestSuite 类的代码如下所示。

```
1 package wyf;
2 import junit.framework.*; //引入相关包
3 public class MyTestSuite{ //声明测试套件类
4     public static Test suite(){ //定义静态的获取测试套件的工厂方法
5         TestSuite ts = new TestSuite("MY TEST SUITE"); //创建测试套件类
6         ts.addTestSuite(MyTestCase.class); //添加测试用例
7         ts.addTestSuite(MyTestCase.class); //添加测试用例
8         return ts; //返回测试套件
9     }
}
```

开发测试套件时，第 3 行定义的 suite 方法是必需的。因为 JUnit 进行测试时，首先会寻找该方法，然后通过其返回的 TestSuite 对象运行其中的全部测试用例。程序通过代码第 6 行和第 7 行添加需要执行的测试用例，该测试套件运行结果如图 13-3 所示。

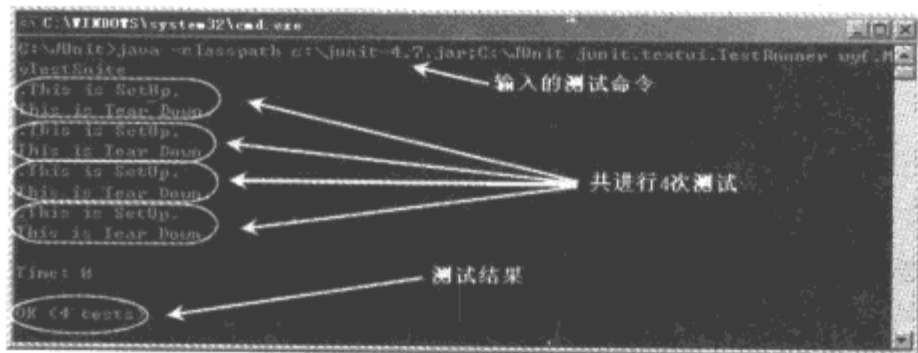


图 13-3 使用测试套件进行测试的结果

**提示** 窗口模式运行测试套件的效果与执行单一 Test Case 测试的步骤类似，本书在此不再赘述。

### 13.1.3 岂曰无衣，与子同袍——JUnit 和 Ant 的联合

说到 JUnit 在 Java 单元测试领域不言而喻的统治地位，不得不提另外一个 Java 在项目构建中的无冕之王——Ant，本节就来对这两个工具做一个强强联合。

“师兄，使用 JUnit 把实际代码和测试代码分开，既方便了管理，还简化了开发，真是好处多啊。”

“呵呵，别着急，JUnit 还有更厉害的呢。”

“哦，那是什么啊？”

“还记得我给你提到过的 Ant 吧，Ant 就可以和 JUnit 很好地配合起来呢。”

在 Ant 中集成 JUnit 的测试模块，可以把对项目的编译、测试和运行彻底整合到一起，对于开发人员来说确实是一个非常受用的事情。下面给出一个简单的例子来说明 JUnit 和 Ant 合作的力量。

本例将继续使用上一小节的 TestMe 类和 MyTestCase 类，不过为了便于使用 Ant 进行项目构建，需要组织如图 13-4 所示的应用目录。

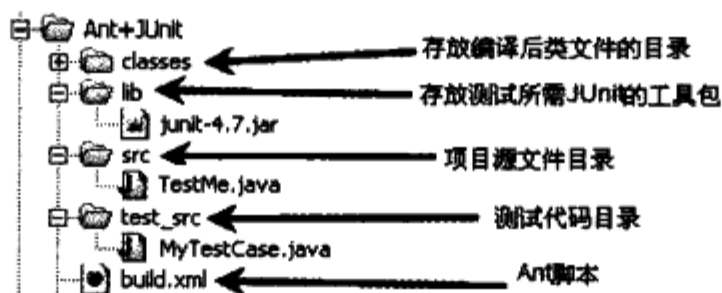


图 13-4 使用 Ant 进行 JUnit 测试的目录结构

组织好如图 13-4 所示的目录结构后，就可以开发对应的 Ant 脚本了，打开 build.xml 文件，在其中输入如下 XML 代码。

```

1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <project name="MyTest" default="Test" >      <!--定义工程名-->
3      <target name="Compile">                  <!--定义 Compile 目标-->
4          <javac
5              destdir="classes"
6              srcdir="src"
7          >                                     <!--指定源文件目录和编译后的文件目录-->
8          </javac>
9      </target>
10     <target name="CompileTest">              <!--定义 CompileTest 目标-->
11         <javac
12             destdir="classes"
13             srcdir="test_src"
14         >                                     <!--指定源文件目录和编译后的文件目录-->
15         <classpath path="lib/junit-4.7.jar"/> <!--指定编译时的 JUnit 的 jar 包目录-->
16         </javac>
17     </target>
18     <target name="Test" depends="Compile,CompileTest"> <!--定义 Test 目标-->
19         <junit
20             printsummary="true"
21             showoutput="true"
22         >                                     <!--设置打印测试结果和程序的输出-->
23         <classpath path="lib/junit-4.7.jar;classes"/><!--设置 classpath-->
24         <test name="wyf.MyTestCase"/>

```



```

25         </junit>
26     </target>
27 </project>

```

最后，打开命令行窗口，将当前目录设置为本次测试项目的目录，然后输入 ant 命令，其运行结果如图 13-5 所示。

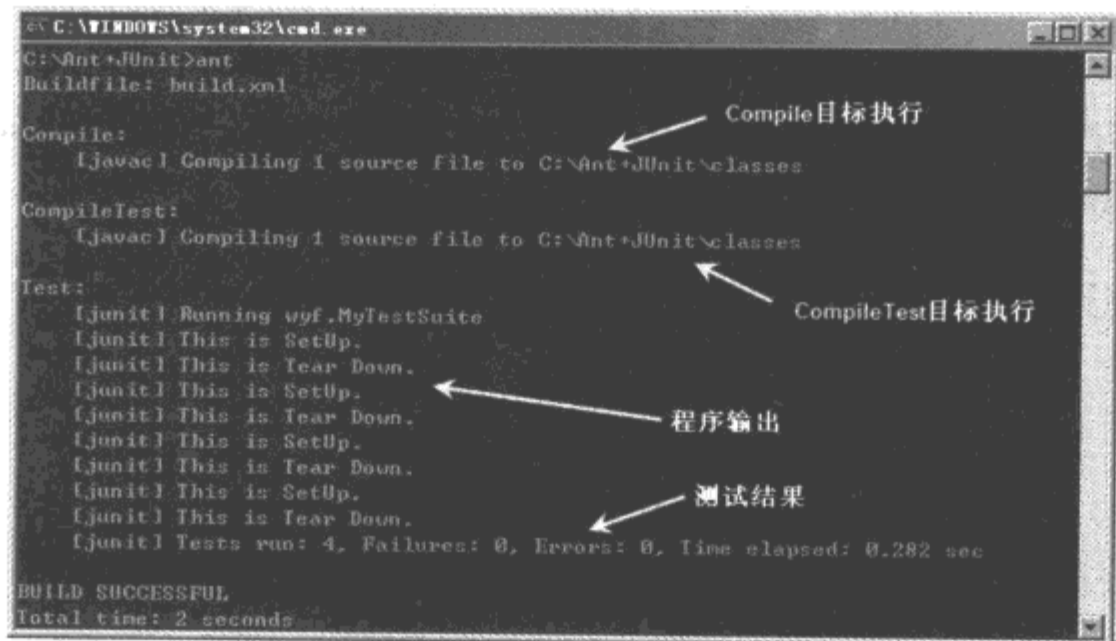


图 13-5 Ant 中执行 JUnit 的运行结果

#### 13.1.4 得道者多助——JUnit 在 Eclipse 和 NetBeans 中的使用

“师兄啊，关于 JUnit 你讲得很不错，不过我们开发的时候用的可是 Eclipse 啊，如何在这些 IDE 中集成 JUnit 的单元测试呢？”

“呵呵，你这个求甚解的学习态度很值得表扬啊，好吧，那我就来向你介绍介绍如何让 JUnit 框架工作在各种 IDE 环境下。”

下面就来介绍如何在市面上比较流行的集成开发环境 Eclipse 和 NetBeans 中使用 JUnit 进行单元测试。本例中只为说明 JUnit 框架的使用，故被测试的类非常简单，如下所示。

```

1 public class Simple {
2     public String sayHi(){ return "Hello"; } //返回一个字符串"Hello"
3     public String sayWrong(){ return "Wrong"; } //返回一个字符串"Wrong"
4 }

```

下面首先在 Eclipse 中对 Simple 类的两个方法进行测试。在 Eclipse 中新建一个项目，在项目中新建一个 Simple 类，输入上述代码。然后在 Package Explorer 中找到 Simple 类，单击鼠标右键，在 New 子菜单项中选择 JUnit Test Case。

在之后的 Test Case 创建向导中，可以根据 Eclipse 的提示决定是否由系统自动创建 setUp、tearDown 等方法，也可以选择需要对 Simple 类的哪些方法进行测试。

创建完成后就可以开发具体的测试代码了，开发细节与前面小节介绍的单独使用 JUnit 的方式相同，本书这里不再赘述。开发完成后在 Package Explorer 中找到 SimpleTest，单击鼠标右键，选择 Run As-JUnit Test。其运行结果如图 13-6 所示。

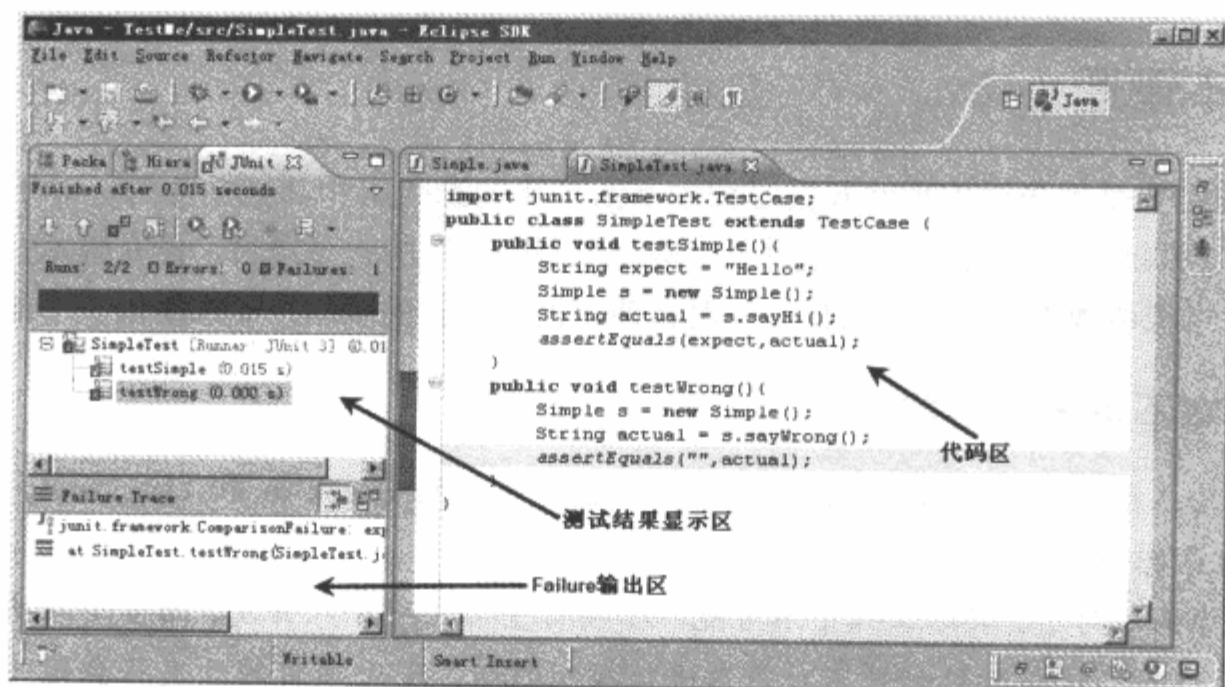


图 13-6 使用 Eclipse 进行 JUnit 测试示例

“呵呵，Eclipse 还不错嘛，可以在创建 Test Case 的向导中将一些琐碎的工作交给 Eclipse 去做，那在 NetBeans 中是怎样的啊？”

“NetBeans 中 JUnit 测试用例的开发与 Eclipse 很相近，IDE 嘛，最主要的目的就是方便开发人员。不过 NetBeans 提供便利的方式并不一样，可以说比 Eclipse 还要快捷。”

在 NetBeans 中新建一个项目，然后与在 Eclipse 中一样开发 Simple 类，之后在项目的测试包下新建一个针对现有类进行测试的 JUnit 测试用例。然后 NetBeans 会提示选择哪个类文件作为该测试用例的测试目标，选择完毕后 NetBeans 会自动生成 Test Case 的大部分代码，开发人员只需要在其中添加测试细节相关部分的代码即可。

假设需要测试的某方法有两个参数，则开发人员只需要对这两个参数和期望的结果进行赋值即可，其他代码都由 NetBeans 自动完成。这比 Eclipse 的 Test Case 创建向导要更加快捷一些。

开发完成测试用例 SimpleTest 之后，就可以运行查看结果了，在 NetBeans 中进行 JUnit 单元测试的界面如图 13-7 所示。

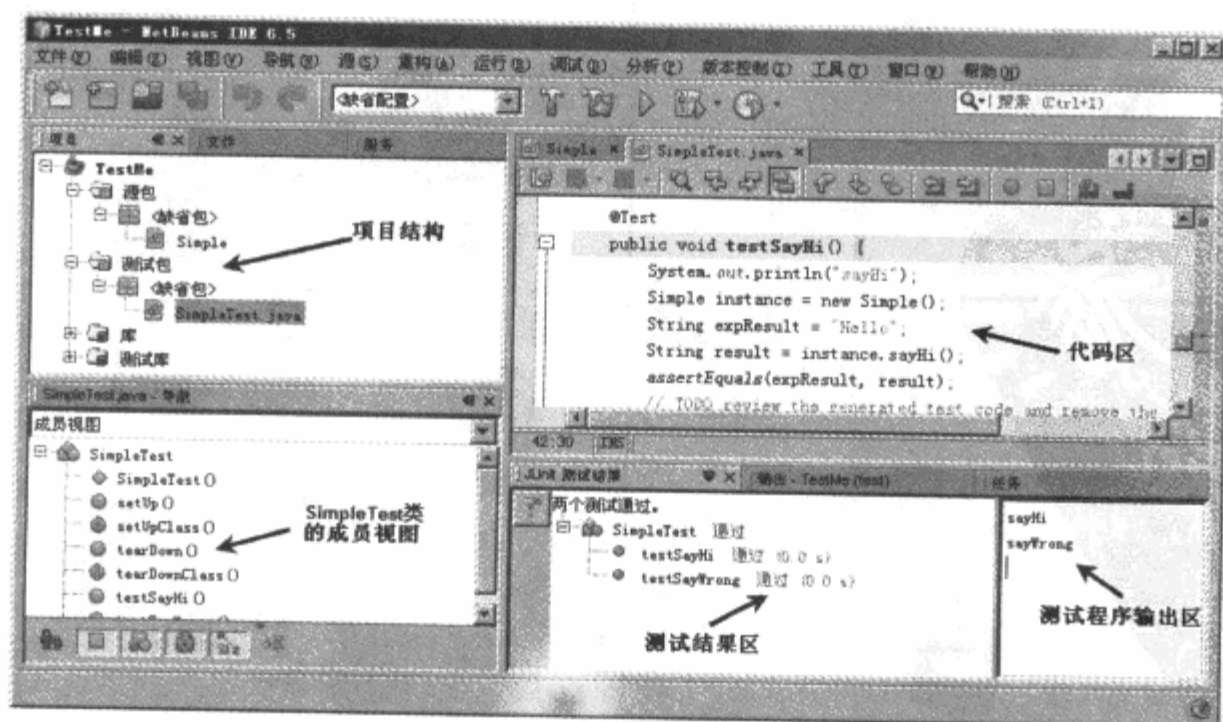


图 13-7 使用 NetBeans 进行 JUnit 测试示例

13.2 版本管理

版本管理主要是用来维护软件开发过程中不断变化的源代码和其他文件的，对于一些小的项目，要求版本控制或许有些过分，但是对于一些需要团队开发的大型项目，没有版本控制将会非常痛苦，本节就来谈谈版本控制的重要性，并对目前市面上流行的版本控制系统做一个简单的介绍。

13.2.1 版本不可一日不控

“哎，师兄，自从上了班以后啊，最不习惯的就是每天的 check-in 和 check-out，好麻烦哦。当初在学校自己想写就写多舒服啊。”

“这就是你的不对了，实际工作中不可能一个项目全部由一个人开发，而多个人协同工作时需要协调的事情很多。这个时候如果不对版本进行控制的话，整个项目的开发将会是一团浆糊。”

在软件项目的开发过程中，最宝贵的就是源代码了，如果不采用版本控制对源代码及其他资源文件进行管理，那么一般就是每个开发人员都有一份当前项目的整体副本各自开发。等到项目最后再将每个人的成果合并到一起变成一个大的项目，这种方式显然弊端多多。

- 不便于集中管理。由于开发人员人手一个项目副本，相当于权力下放，而且也没有一个主要的版本作为项目的核心，这样管理起来就太麻烦了，而且项目开发完成后需要花费相当多的精力去将各个副本融合成最终的版本。
- 代码无法及时更新。如果没有版本控制，代码之间的更新也就成了问题。例如某一个副本中对于一个方法的代码进行了修改，而其他副本并不知道，仍然按照原有的方式去调用这个方法，如果这个方法的使用频率非常高，那结果会错得越来越离谱。

软件项目的这种分散单一式的管理对于项目的开发非常不利，而采用版本控制则可以消除这些缺陷，二者的比较可以用如图 13-8 所示的漫画来表示。

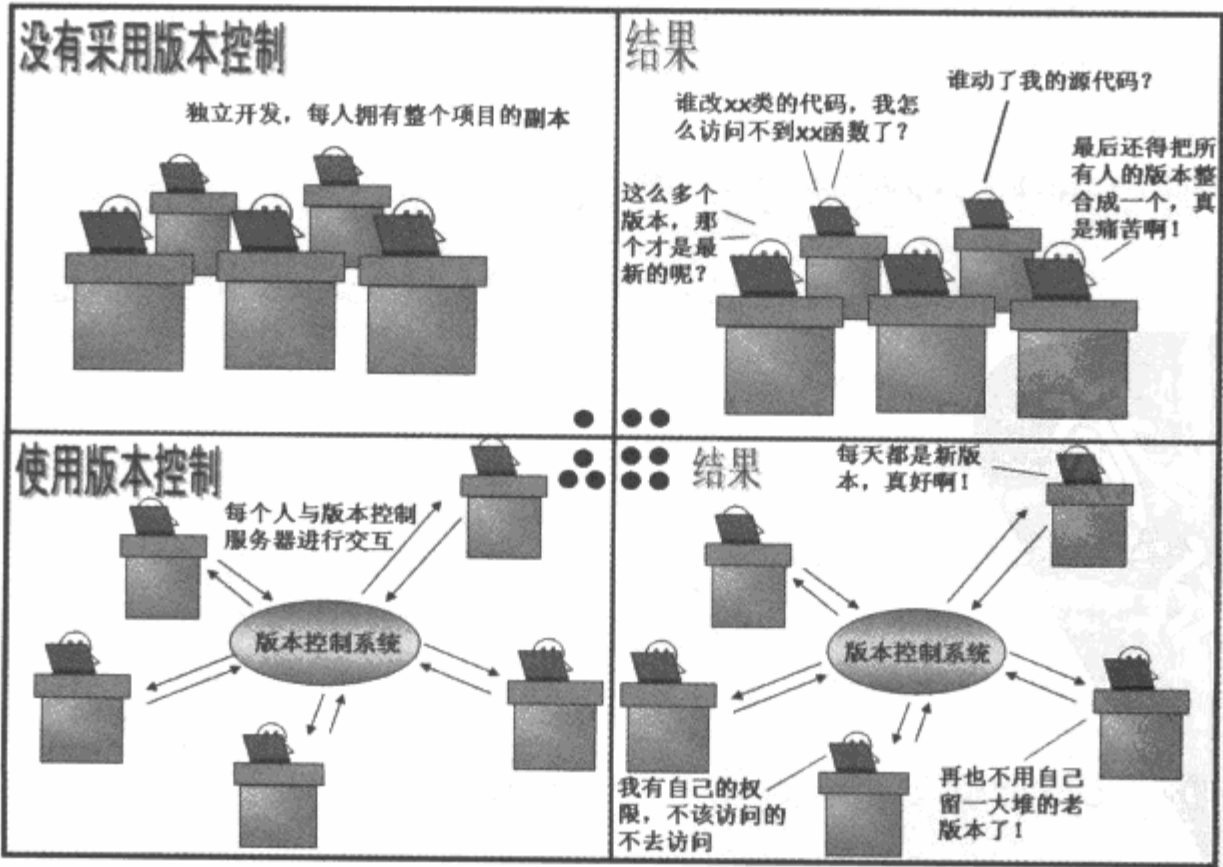


图 13-8 是否采用版本控制系统的对比

版本控制存在于项目的整个生命周期中，主要负责对项目进行宏观调控，对项目的源代码等资源版本进行集中管理，以及消除代码冲突等任务。好的版本控制是项目开发顺利进行的一个重要保证。

在版本控制中，项目的所有版本统一保存在版本控制服务器中，开发人员需要从版本控制服务器中下载需要进行开发的版本，开发完成后再将其上传到版本控制服务器，这里面下载和上传分别叫做 check-in 和 check-out。这就大大集中了管理，减少了版本的过分冗余。

“版本控制在开发中非常有必要，例如在开发过程中会对多个项目版本进行维护，当某一个版本出现一些难以排除的错误时，就可以参照之前的版本进行找错，这样效率会高很多。”

“嗯，的确。万一搞不定了就像数据库那样直接回滚，退到之前的版本。”

“呵呵，差不多就是这样，同时需要注意的是在软件测试的时候，如果不对项目的测试版本进行清晰的管理，那么往往会延长测试的时间。”

“看来需要版本控制的地方还真不少啊！”

处理开发人员对代码的操作时，不同的版本控制系统有不同的策略，主要分为集中式和分散式。集中式是指版本中的某段代码（通常是一个类），在同一时刻只能被一个开发人员 check-in，对其访问并操作，相当于为其加上了锁。而分散式策略则是在同一时刻每个开发人员都可以自由地对整个代码进行访问和编写，如果在开发人员 check-out 的时候发现有两个或多个开发人员都对某段代码做了修改，这就是产生了冲突，任何支持分散式策略的版本控制系统都会采用一些手段来消除冲突，融合冲突代码。

### 13.2.2 沙场秋点兵之版本控制系统

“师兄，既然版本控制系统这么重要，你就给我介绍介绍 CVS 的用法吧。”

“等等，我怎么听着这话不对劲啊，你是不是以为 CVS 就是版本控制系统啊？”

“啊，不是吗？CVS 不就是版本控制系统的缩写吗？”

“当然不是啦，目前市面上流行的版本控制系统有很多，CVS 只是其中的一个，况且 CVS 也不是版本控制系统的缩写。”

版本控制系统有很多，本小节将选择 Java 开发中常见的几个版本控制系统做简单的介绍。

#### 1. CVS

CVS 的全称是 Concurrent Versions System，是比较早的一种版本控制系统，也是使用比较广泛的版本控制系统之一。CVS 是基于 C/S 架构的，项目的源代码等资源文件都存储在服务器中。所有需要的信息都从服务器获得，所以不同地点的开发人员也可以同时进行开发，这就是所谓的分布式团队。

CVS 的服务器中保存有整个项目的代码，但并不是为所有版本都进行备份，而只是保存不同版本之间的差异，这样可以节省空间。使用 CVS，不仅可以对项目的版本进行管理，还可以对不同版本的代码进行对比等操作。

任何版本管理系统都需要解决代码冲突的问题，目前 CVS 的一个比较吸引人的特性就是采用不对文件加锁的分散式管理方式。假设开发人员需要对服务器上的 A 类进行修改，首先从服务器

上 check-in 这个类的代码进行修改，而与此同时另外一个开发人员也可以 check-in 这段代码进行修改，CVS 会在提交的时候通过自动方式或者通知开发人员来解决这个冲突问题。

## 2. Mercurial

Mercurial 是一款比较年轻的版本控制系统，于 2005 年发布，Mercurial 是由 Python 和 C 语言编写的，其最初被设计用来替代 BitKeeper 进行 Linux 项目的版本控制。后来也发展出了 Windows、苹果等其他操作系统平台的版本，Mercurial 还可以提供基于 Web 界面的版本控制。

Mercurial 对于版本控制的性能非常重视，所以虽然 Mercurial 小巧，但是功能却很强大，它采用了比较快速的方法来对版本进行修订等操作，处理文本的效率也非常高。同时为了与服务器进行快速通信，Mercurial 还开发了一套自己的通信协议。

Mercurial 目前主要应用在 Open Solaris、Mozilla、Linux 等项目中，同时 Java 的开发环境 NetBeans 也支持使用 Mercurial 作为版本管理系统。

## 3. Subversion

Subversion 又称为 SVN，其研发者对于 Subversion 的期待就是逐步取代 CVS。既然有这种说法，那么 Subversion 肯定就有其出色的地方。Subversion 除了继承了 CVS 管理源代码的优良特性之外，还摒弃了 CVS 的一些弊端，如不能很好地处理 Unicode 文件、不支持文件的移动等。

Subversion 中还增添了许多新的特性，如为所有文件提供统一的版本号，不管在一次修改中某个文件有没有被改动，都会更新其版本号。另外，Subversion 还提供了对项目文件进行追踪等功能。Subversion 也集成了 Web 应用，客户端可以通过 Web 方式访问版本控制系统。

从诞生至今，Subversion 慢慢获得了原本使用 CVS 的广大开发人员的好感，目前如 Apache Software Foundation、Gnome、FreeBSD 等项目已经逐渐从 CVS 转换到了 Subversion。作为一款更先进的版本控制系统，Subversion 在未来必将受到大部分开发人员的青睐。

### 13.2.3 版本控制系统与 IDE 的协作

“哎，师兄，版本控制这个话题谈起来还真有意思啊。”

“呵呵，在介绍了这些版本控制系统之后，照例我们来看看 Java 开发中一般采用的版本控制系统，说白了，就是看看我们的 IDE 中都支持什么版本管理系统。”

下面同样针对 Java 开发界的两大 IDE 分别介绍其在版本控制系统方面的特点。

#### 1. Eclipse

Eclipse 中的版本控制主要是基于 CVS 的，即 Eclipse 中提供了 CVS 版本控制系统的客户端，开发人员只需要创建或连接 CVS 服务器即可。如在一个项目中可以通过右键单击项目选择 Team 菜单项中的 Share Project 选项，就可以创建一个代码的仓库（Repository）。默认情况下 Eclipse 中是不显示 CVS 的 Repository 的，可以通过 Window 菜单的 Show View 将其显示，如图 13-9 所示。

由于 CVS 已经慢慢老去，开发人员在 Eclipse 中越来越多地使用到其他的版本控制系统，利用 Eclipse 插件的优势，添加其他版本控制系统也非常容易。目前市面上流行的版本控制系统（如 Subversion 等）都可以通过安装插件的方式与 Eclipse 集成。





图 13-9 Eclipse 中的 CVS 代码仓库

## 2. NetBeans

相比于 Eclipse, NetBeans 中将版本控制系统做得更加到位, 通过预先配置, NetBeans 默认支持三种版本控制系统: CVS、Mercurial 和 Subversion, 这样基本满足了各种开发的需要。在 NetBeans 中, 可以在窗口菜单中找到版本控制子菜单并选择需要显示的版本控制系统, 如图 13-10 所示。

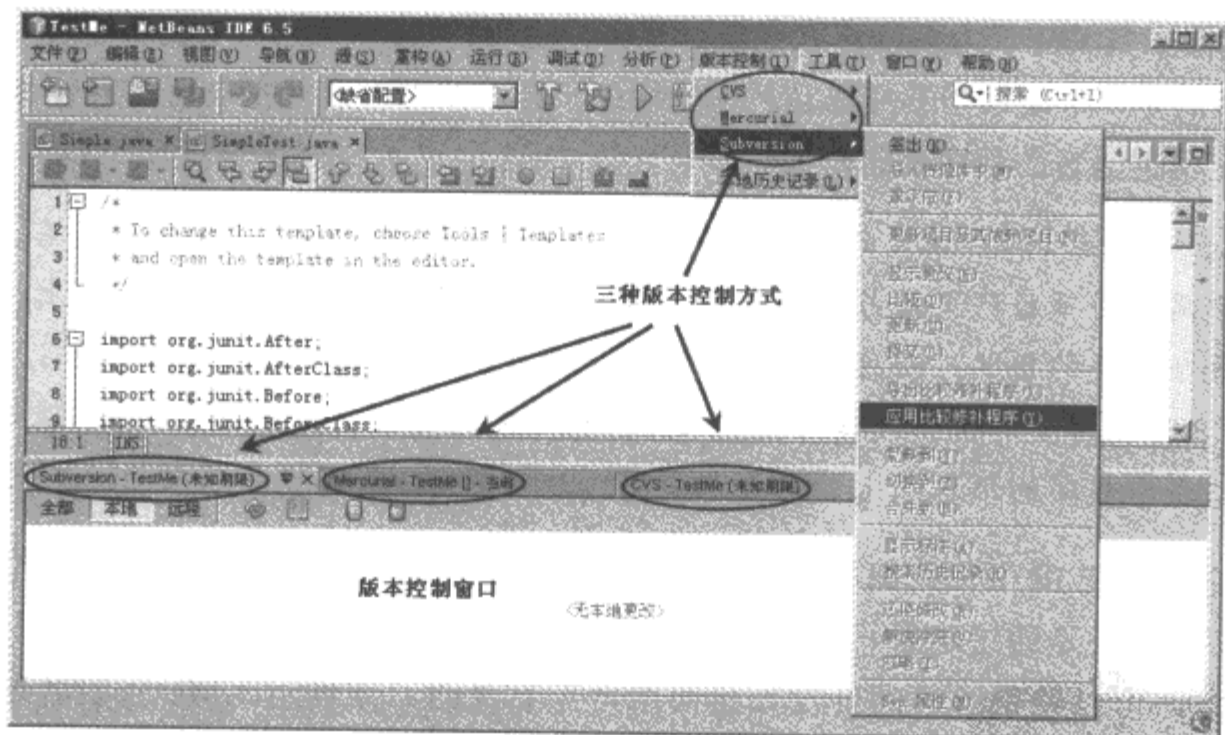


图 13-10 NetBeans 中可支持的版本控制系统

在每种版本控制方式中, NetBeans 都提供了一些诸如代码行比较、更新远程版本控制目录等多种功能。如果需要以上三者之外的版本控制系统, 也可以通过安装插件的方式进行补充。

## 13.3 UML 建模语言

在一个实际项目的开发过程中, 并不是从头至尾都是在敲代码。恰恰相反, 整个过程中真正编程的时间或许不到一半, 更多的时间和精力会放在对系统的分析和设计上。采用 UML 建模方式

可以对系统做出更好的分析和高效的设计。本节将对 UML 建模语言做一个简单的介绍，同时介绍如何在 Java 的开发环境中通过使用插件来建立 UML 模型。

### 13.3.1 UML 就这么回事

“师兄，我前两天去书城淘书，买了一本讲 UML 的书，我当时以为是跟 XML 类似的新语言呢，结果回来后大眼瞪小眼地看了半小时，然后就被我束之高阁了。”

“呵呵，UML 的确不属于 HTML、XML 家族，不过 UML 对于开发人员，尤其是高级开发人员来说，也是个软件设计时必不可少的好帮手呢。”

“嘿嘿，看来我还没买错哪，我很有远见嘛。师兄，你已经是个非常高级的开发人员了，那你给我唠唠 UML 是怎么回事吧？”

UML 的全称是 Unified Modeling Language，它提供了一种对软件系统进行分析和设计的方法，其借鉴了如 Booch、OMT 之类的优秀软件工程思想。虽然 UML 并不是市面上唯一的建模语言，但是作为工业标准，UML 是功能最强大的。

在软件开发中，所面临的困难和挑战主要有两个：一个是越来越复杂的系统结构，另一个就是不断变化的需求。两种情况下都需要对系统进行合理的设计，使它能够通过可行的开发过程来实现复杂的系统功能，同时还能具有强壮的结构以应对不断变化的需求。这时候就需要使用 UML 来对系统进行抽象和模型化了。

UML 建模工具关注于为系统建立一个可视化、说明性强的模型。UML 并非基于某一种编程语言，它提供的是一个凌驾于编程语言之上的设计平台。在模型建立完成后，UML 建模工具可以自动生成指定编程语言的程序代码框架。

“同时呢，我觉得 UML 最大的好处就是提供了一种供软件开发人员交流的语言，你想想看，假设在开发中需要将项目移交给另一个开发人员，如果只是将密密麻麻的代码奉上，他势必无法弄清楚其中每个模块的含义，而可视化的系统模型则可以让他迅速地熟悉整个项目。”

“嗯，师兄你说得太对啦！代码实现了人与机器之间的交互，而 UML 可以实现人与人之间的交互。师兄，我看我买的那本书上面好多都是图，这个 UML 是否就是用来画图的啊？”

通俗地讲，UML 的可视化编程就是通过各种图来体现的，但进行 UML 建模绝不仅仅是简单地画图，UML 正是通过这种可视化编程，描述了整个应用系统的模型结构。一个通过建模抽象好的模型，可以使得随后的开发如鱼得水，左右逢源。下面对 UML 中的各种图做一个简单的介绍。

UML 中的图都是由一些基本元素构成的，这些基本元素有的代表与编程无关的含义，如用户等，有的则代表编程中的一些实体，如 Java 中的类、接口等。

图是 UML 中主要的建模原材料，UML 中的图可以分为行为图和结构图。其中行为图包括顺序图、状态图、活动图、协作图等；结构图包括类图、对象图、构建图、部署图等。图 13-11 给出了一个类图的示例。

类图主要描述类、接口、包之间的关系。图 13-11 中描述了这样一个模型：在 wyf 包下面有两个类和一个接口。类 Vehicle 中定义了一些有关汽车类的属性和方法，而变形金刚类 Transformers

继承自 Vehicle，同时由于变形金刚可以变形，于是实现了接口 Transformable 中的 transform 方法。这是一个比较简单的类图模型。

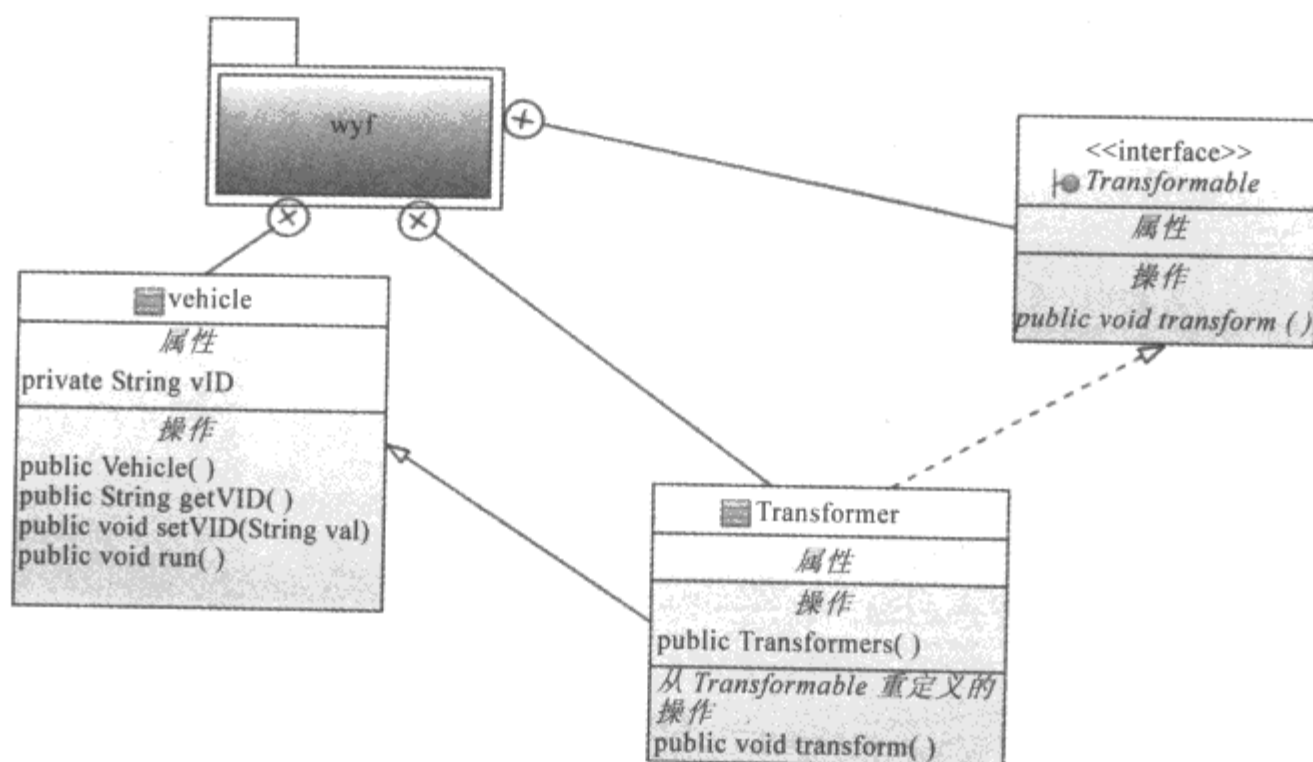


图 13-11 UML 类图示例

多个图组合起来就称为视图，视图是对整个系统模型在某一个角度上的横切，整个系统模型就是由多个视图组成的。

### 13.3.2 UML 之实战 IDE

“师兄，我比较笨，你讲来讲去我还是不能彻底明白 UML 到底是怎么工作的，老规矩，来点示例吧，嘿嘿。”

“就知道你会这么说，好吧，咱们今天就来搞一搞建模，体验一下 UML 的功能。”

市面上的 UML 建模工具有很多，如 IBM 的 Rational Rose、微软的 Visio 等。同时还有很多免费的 UML 工具，在 java-source 网站上面可以找到很多免费的 UML 工具如 StarUML、ArgoUML 等。但是比较受欢迎的方式，还是在集成了 UML 工具的 IDE 中进行建模。

本小节就以 NetBeans 为例，通过在 NetBeans 中使用 UML 建模工具可视化编程开发一个小项目来说明这个问题。在开发之前首先需要下载与 UML 相关的插件，在 NetBeans 中下载安装插件比较容易，只需要在 NetBeans 的插件列表中选择需要的插件，NetBeans 会自动下载安装。

本例中首先新建一个空项目 MyVehicle，然后新建一个基于 Java 平台模型的 UML 项目 MyUML，在图类型向导中选择类图。接着在新建的类图中构造曾在前面小节中提到的有关变形金刚的类图（如图 13-11 所示）。

UML 建模完成之后，就可以在 UML 项目上单击鼠标右键选择自动生成代码了，在自动生成代码时需要选择目标项目，即将新生成的代码放到哪个项目中，本例中目标项目为 MyVehicle。类图所对应的代码框架将会自动生成到 MyVehicle 项目中。整个 NetBeans 的 UML 建模界面如图 13-12 所示。

在 Eclipse 中使用 UML 工具进行建模的方式与在 NetBeans 中类似，只是插件的安装有区别罢了，本书由于篇幅有限，不再举例，感兴趣的读者朋友可以自己尝试。

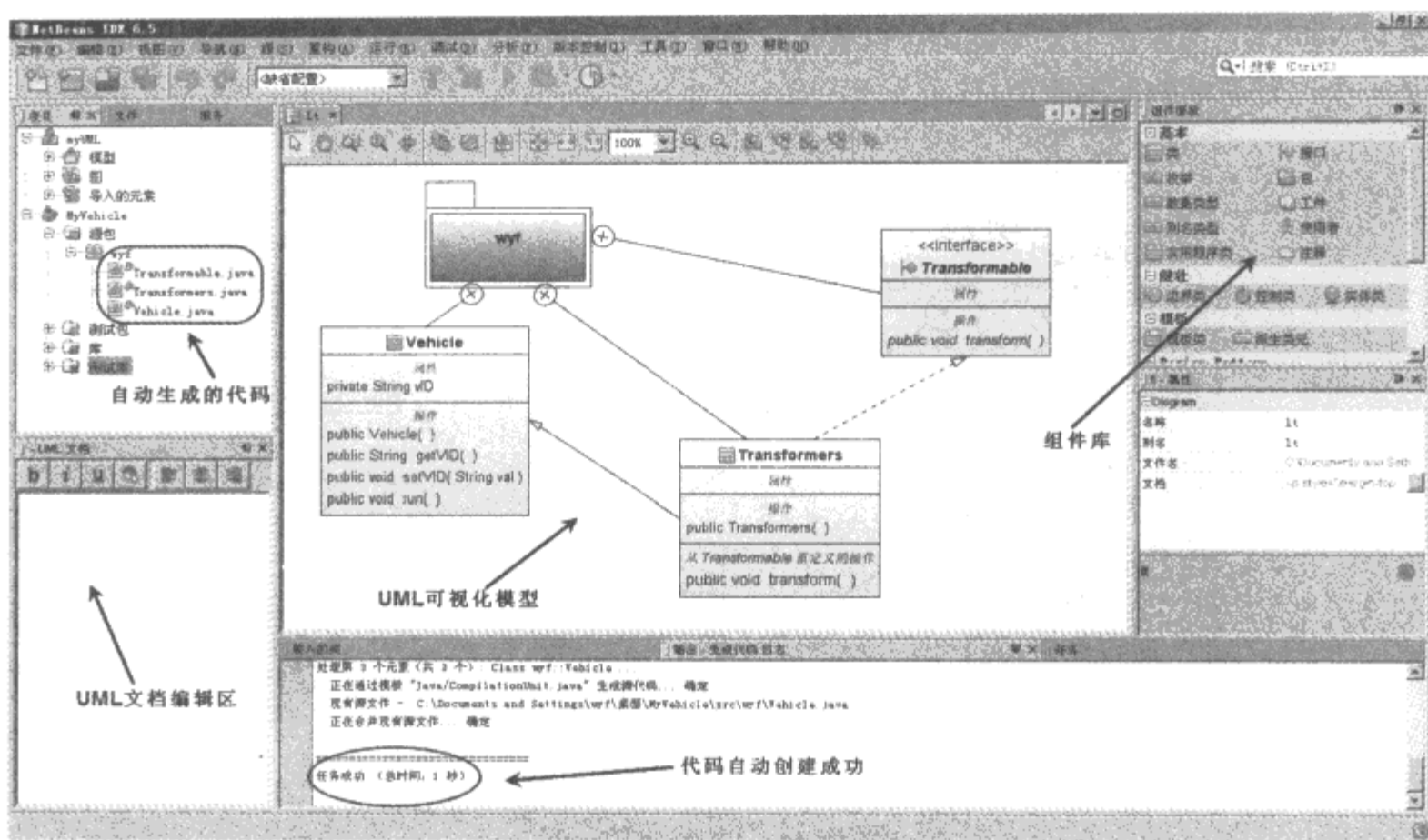


图 13-12 在 NetBeans 中通过 UML 创建项目

## 13.4 大型服务器操作系统

毋庸置疑，各位读者将来会在工作中参与到许多项目的开发，而这些项目中没有几个是需要部署到个人电脑上的。Java 的企业级应用程序几乎都是部署在大型服务器上的，具体来说就是大型服务器的操作系统上。于是为了自己以后能在部署项目时不会手忙脚乱，不妨在本节先与各大型服务器操作系统混个脸熟。

“哎，师兄。好后悔啊，当初在学校的时候没有多研究研究其他的操作系统，结果到了工作中才发现，原来 Java 开发出来的项目部署的地方全是那些大型的服务器级操作系统，而且系统的操作方式跟个人电脑上的一点都不一样呢。”

“呵呵，后悔也没有用啊，不如多抽出些时间去学习学习它们。”

“我也想啊，那师兄你给我介绍一下目前的服务器操作系统都有哪些吧，我学习的时候也好瞄准位置，一枪毙命。”

目前针对那些部署 Java 项目的服务器操作系统大致可以分为 UNIX 平台、Linux 平台和 Windows 平台，每终平台下也有不同的具体实现，下面就来对这些服务器操作系统做一个简单的介绍。

### 13.4.1 UNIX 平台

UNIX 是一个多任务、多用户的操作系统，其诞生于 20 世纪 70 年代的贝尔实验室，UNIX 是用 C 语言编写的，其在商业上有多种实现。UNIX 是大型服务器操作系统中市场占有率较高的一个平台，各种主流 UNIX 版本的 logo 如图 13-13、图 13-14 和图 13-15 所示。



图 13-13 Solaris 系统 logo



图 13-14 IBM AIX 的 logo



图 13-15 HP-UX 的 logo

### 1. Solaris

Solaris 的 logo 如图 13-13 所示, Solaris 原名 SunOS, 是 Sun 公司推出的一款 UNIX 操作系统。Solaris 系列有 Solaris 和 OpenSolaris, 前者为商用, 目前最高版本为 Solaris 10, 后者是由 Sun 在 2005 年开放的正在研发的 Solaris 11 而来的, 是开发版本。

Solaris 一般运行在 Sun 的工作站上, 其硬件架构主要为 SPARC 系列, 不过 Solaris 也支持 X86 系列的硬件。在 UNIX 的各商用版本中, Solaris 由于性能不错而价格也较低, 用户群很大。

### 2. IBM AIX

IBM AIX 的 logo 如图 13-14 所示, 它是由 IBM 开发的 UNIX 操作系统。IBM 主要关注于 AIX 在商业上的应用, 所以具有很多优于其他 UNIX 版本的新特性, 如虚拟服务器、集群管理和安全性等。2007 年推出的 AIX 6 是目前最高版本, 其在虚拟化和安全性方面更加出色。

与 Solaris 一样, IBM 的 AIX 首先可以运行在 IBM 自己的工作站如 RS/6000 上, 也可以运行在一些其他的大型并行计算机上。

### 3. HP-UX

HP-UX 的 logo 如图 13-15 所示, HP-UX 是惠普公司开发的 UNIX 操作系统, 其全称为 Hewlett Packard UNIX。HP-UX 是 UNIX 的另一个变种, 目前最新的版本为 HP-UX 11i。

似乎能够开发基于 UNIX 标准操作系统的厂商都有能力做支持自己操作系统的工作站及硬件, 惠普也不例外。HP-UX 主要运行于自己研发的 PA-RISC 硬件平台上, 同时也可以运行在安装了 Intel Itanium 处理器的服务器上。虽然用户群体没有 Solaris 众多, 但 HP-UX 在电子商务领域也有着不错的口碑。

## 13.4.2 Linux 平台

Linux 诞生于 UNIX 之后, 由著名的程序设计大师 Linus Torvalds 开发, 后来经过很多人的努力而越发强大。Linux 从诞生至今一直是开源免费的, 最主要实现版本的 logo 如图 13-16、图 13-17 和图 13-18 所示。



图 13-16 RedHat 的 logo



图 13-17 SUSE 的 logo



图 13-18 Ubuntu 的 logo



### 1. RedHat

RedHat 的 logo 如图 12-17 所示。RedHat 可以说是 Linux 界的老大哥了，它占据了 Linux 系统平台的半壁江山。RedHat 桌面版的最终版本为 RedHat 9.0，从那之后 RedHat 一直将重心放在 RedHat 在企业级中的应用。

RedHat 是国内最受开发人的喜爱的 Linux 操作系统之一，也是很多初学者或在校学生学习 Linux 操作系统的一个不错选择。本书前面曾提到的 RHCE（Red Hat 认证工程师）就是 Linux 最为权威的认证。

### 2. SUSE Linux

SUSE Linux 的 logo 如图 13-17 所示，其原本是德国一家公司开发并维护的一款 UNIX 操作系统，后来该公司被 Novell 收购。Novell 是一家资格比较老的网络公司，它在收购 SUSE 之后为其注入了一些企业级应用和桌面应用的新特性。

SUSE 系列的 Linux 有多个版本，除了企业级版本 SUSE Enterprise Linux Server，还包括零售版本和开发源代码版本等，SUSE 较其他 Linux 的一个明显特点就是在用户界面上做得比较出色。

### 3. Ubuntu

Ubuntu 原是传达了一种生存理念的非洲方言，在这里代表的是一款 Linux 系列的操作系统。Ubuntu 的 logo 如图 13-18 所示，从其 logo 中也可以看出 Ubuntu 蕴含着分享的思想。

Ubuntu 着重于系统安全性和易用性，其每隔 6 个月就会更新发布一个较新的版本，Ubuntu 有着非常繁荣的开发者社区，这也是其能够迅速流行的原因之一。

## 13.4.3 Windows Server 平台

Windows Server 是微软的服务器级操作系统中的一个大家族，其最新的版本为 Windows Server 2008，logo 如图 13-19 所示。其在灵活性、安全性、文件存储等方面做了很大的改进。



图 13-19 Windows Server 2008 的 logo

Windows Server 基于微软深厚的操作系统研发功底，使其在很多地方既有着网络操作系统的强大，又有着个人 PC 般的简易友好，非常适合中小企业用于快速搭建网络服务器。

## 13.5 集群与负载均衡

项目开发完成，测试结束后，还不能说开发人员的工作就此结束了，因为还需要将项目部署到服务器中。对于一些小型项目，部署是个比较轻松的过程，但是对于一些负载大且对性能要求较高的系统，就必须采取一些集群的策略了，本节就来向读者介绍在部署大型应用系统时的一些相关知识。

### 13.5.1 集群

“师兄，像你们这样的高手，现在每天除了写代码，应该有其他的事情操心吧？”

“的确啊，现在代码写的是不多了，不过做其他的也很累。”

“哦，那都是做什么啊？”

“比方说搭搭集群啦、研究研究负载均衡啦。”

“是吗？那你给我说说呗，让俺也窥视一下这些高级技术活。”

集群，顾名思义，就是将一组计算机集中到一起的意思，这里的计算机不仅指硬件，也包括软件。集群的用途很广泛，比较出名的网格计算就是基于集群的。但在本节中，集群的主要作用还是为了让 Java 开发的项目具有更好的性能和可靠性。在一般情况下，部署项目搭建的集群可以分为以下两种。

- 双机集群

双机集群是将两台服务器捆绑到一起，这两台服务器可以是主从关系，即某一台为主服务器负责处理来自客户端的消息。另外一台为辅助服务器，平时不工作，只在主服务器由于故障瘫痪掉时才会接替主服务器转入到工作状态，这是比较常见的热备份方式。

双机集群中的两台服务器也可以独立地运行不同的程序，只是在某一台瘫痪掉后另一台在继续自己工作的同时，兼任瘫痪掉服务器的职责，这种备份方式与前一种比起来对于资源要节省很多，尤其是在两个服务器瘫痪的几率非常小时这种搭建方式比较经济高效。

- 多机集群

多机集群是将多个服务器组织到一起，每台服务器完成相同的工作。在多机集群中，每台服务器都可以作为其他服务器的备份，这样只要集群中尚有一台服务器存在，这个系统就能够为外界提供服务。通过搭建多机集群，也可以实现整个系统的 HA（High Availability，高可用性）。

“看来系统在真正运行的时候，需要考虑的东西还真多啊。”

“那是当然，服务器必须随时保持可用且高效，否则服务器提供服务的目标客户就会抱怨，一旦流失客户，整个服务器也就没有价值了。”

“嗯，是啊，采用集群能够很好地实现 HA，可是高效是如何实现的呢？”

“要想实现服务器的高效，就必须采取负载均衡技术了。”

负载均衡技术是搭建了集群，尤其是多机集群之后必须要做的一件事。在一个集群中实现负载均衡需要使用到负载均衡器，当客户端发来与服务端连接的请求时，负载均衡器会根据集群中所有服务器的负荷状态，采用特定的算法给用户指派一台负载较低的服务器，其过程如图 13-20 所示。

集群中负载均衡器可以将作业均匀地分布到每一台服务器中，这样每一台服务器的处理能力都会被充分利用到，不会因为一台或几台服务器的崩溃或繁忙而导致系统性能的下降。所以说服务器的集群既保证了系统的高可用性（HA），也大大地提高了系统的性能。



**■ 提示** 负载均衡也可以由软件方式实现，也可以通过硬件方式实现。负载均衡技术及集群的搭建技术所涉及的知识很多，本书将不再赘述，感兴趣的读者可以自行查阅资料学习。

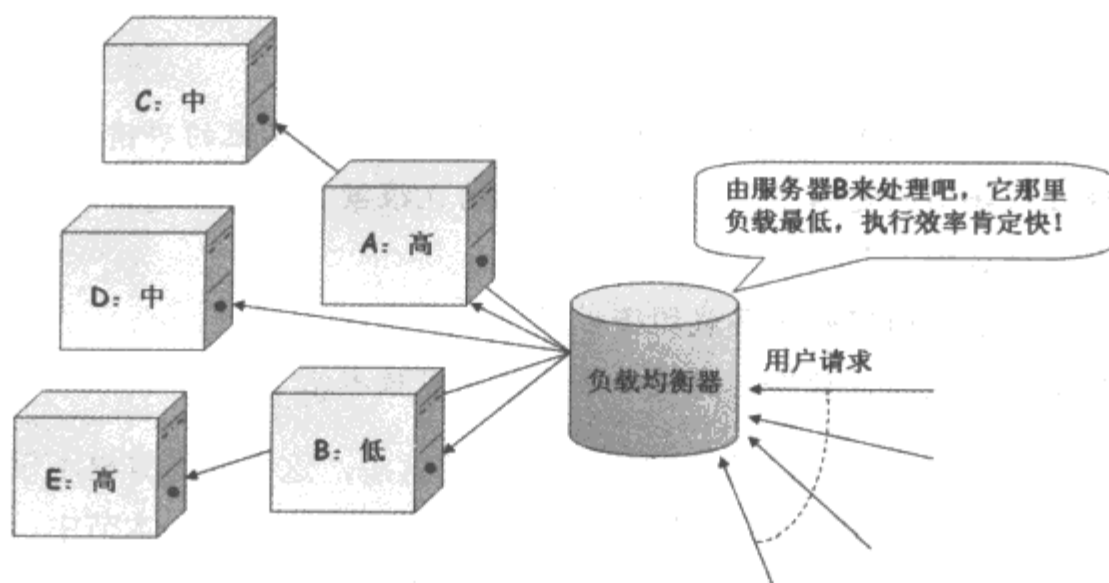


图 13-20 均衡负载器处理过程

### 13.5.2 幂等操作

“哎，蔡佳娃，我这个人向来做事做到底，既然我们提到了使用集群可以实现高可用性，就必须来提提幂等操作了。”

“幂等操作？好生疏的一个词啊。”

“呵呵，看来你把大学时期的好多课程都忘得一干二净了啊！”

在离散数学中，有等幂元这样一个概念，等幂元是指在代数系统中，在等幂元上施加的  $n$  次可结合运算，其结果总是等幂元自身，如在乘法的代数系统中，1 就是等幂元。本节中要提到的幂等操作和等幂元也有一些相似之处。

前面已经提到过，集群的一个重要作用就是实现系统的高可用性，即某一台服务器崩溃以后，可以由后继的服务器接替其继续工作。例如用户一直在和集群中的 A 服务器进行通信，突然 A 服务器瘫痪了，集群中会立刻有一个服务器站出来继续为该用户服务，使用户并不会感觉到系统内部的变化，如图 13-21 所示。

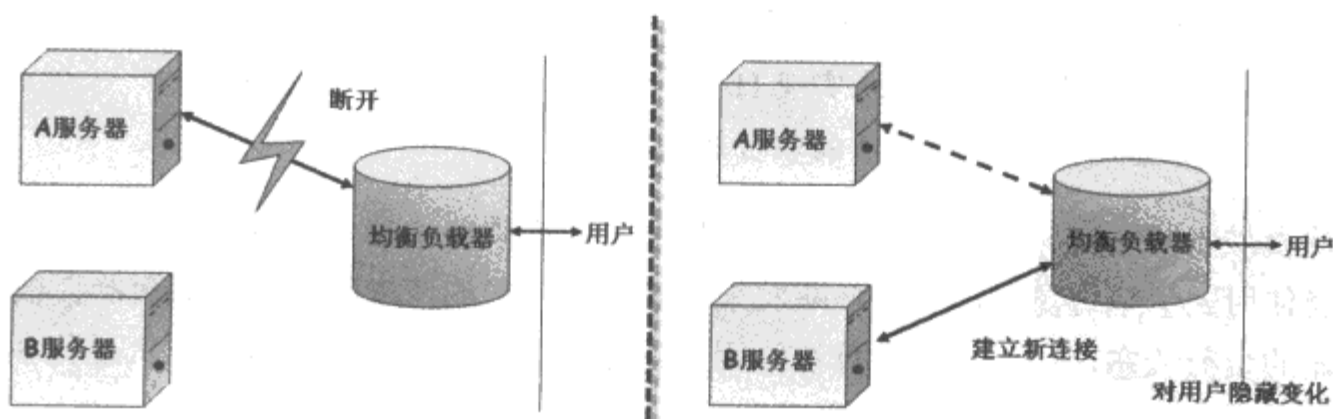


图 13-21 集群高可用性的体现

这种方式看起来很好，但也带来了一个问题，那就是原先在 A 服务器上进行到一半的事务，或者叫 Session 该如何处理？丢弃还是将其迁移到 B 服务器呢？

- 不进行 Session 迁移

如果不进行 Session 迁移，那么当用户的连接活动转接到 B 服务器上时，所有的操作需要从

头再做一遍。这种方式有一定的合理性，但是如果用户在 A 服务器上已完成的操作中涉及一些钱款的转入转出的话，从头再做一遍势必会造成一些糟糕的后果。全部回滚后虽然没有严重后果，但也很麻烦。

#### ● 进行 Session 迁移

进行 Session 迁移是指在 A 服务器崩溃的瞬间，会立刻将 Session 转出给 B 服务器，B 服务器在接替 A 服务器和客户交互时，也会获得 A 服务器的“遗言”——用户的 Session，这样一来就免去了从头再来的苦恼。

“师兄，看来 Session 迁移可以很完美地解决这个问题呢。”

“非也，虽然经过 Session 迁移，但是迁移后并不一定能精确地从上次断开的地方继续执行，只能记录到死在了哪一个操作上，然后从该操作入手，将该操作整个执行一次，再继续执行下面的操作。”

“哦，这样啊。那如果断开的那个操作中已完成部分也涉及了钱款等重要数据的转入转出，那不就不和不进行 Session 迁移一样了吗？”

这种情况下就需要采用幂等操作了，幂等操作是这样一种操作，对于同一种状态，进行多次幂等操作之后，产生的结果是相同的。例如在上面的例子中，假设在某一操作中 A 服务器崩溃，而该操作已经执行过了转账这个环节，那么 B 服务器在收到迁移过来的用户 Session 并继续执行该操作时，如果断开的操作是幂等的，那么在该操作的恢复执行中不会再次进行转账，这就非常好地解决了 Session 迁移所带来的弊端了。



**显示** 幂等操作远非本节例子中描述的那么简单，幂等操作的实现是有一定困难的。一般情况下除了银行等对数据准确性要求高的企业，幂等操作还是比较少用的。但是如果在集群系统中实现了 Session 迁移，就非常有必要实现幂等操作了。

### 13.5.3 我们的程序运行在哪

“师兄啊，我们谈了程序怎么开发，谈了程序运行的操作系统等好多内容，那么最终我们的程序是运行在什么鬼地方呢？”

“呵呵，好吧，我们今天就来谈谈让我们熬夜奋战开发出来的程序最后到底跑到哪里去了，了解这个对于你以后设计和部署大型应用系统也会有所帮助的。”

承载软件的实体当然是硬件，这些运行大中型应用程序的硬件一般都放置在机房中，机房为这些硬件提供必要的电能、温度等条件以使其正常工作，这有些类似于藏酒的酒窖。对于一些大型企业，一般都有自己的机房，但是为数较多的中小型企业一般是把自己的硬件托管到其他机房的。

下面具体说说程序运行在什么硬件上。Java 面向企业级的应用较多，一旦到了要进机房的地步，其硬件设备就应该是小型机或服务器，在前面介绍大型服务器操作系统中提到过的 IBM 和 Sun 的服务器都属于这个范畴。

但是随着前面提到过的集群技术的发展，人们发现将 PC 服务器搭建成集群将会比相同成本下购买的小型机性能高出很多，综合下来，廉价的 PC 服务器集群也就慢慢开始繁荣起来。

“哦，原来在机房中还会看到我们每天朝夕相处的 PC 机箱啊，好亲切哦。”

“不过你可不要误会了，在机房中的 PC 服务器一般可不是我们传统家用的塔式机箱哦。一般的服务器集群都是放置在托管机房中，而托管机房的收费标准首先就是体积，你不觉得我们的塔式机箱有些太碍眼了吗？”

“哦，那是什么样的 PC 服务器啊？”

“这些 PC 服务器叫做机架式服务器，它们所占的空间较小，就像抽屉一样可以抽拉。”（见图 13-22）

“那机架式服务器是最小的吗？”

“不，还有一种更紧凑的叫刀片服务器，不过一般需求下使用机架服务器就可以了。”

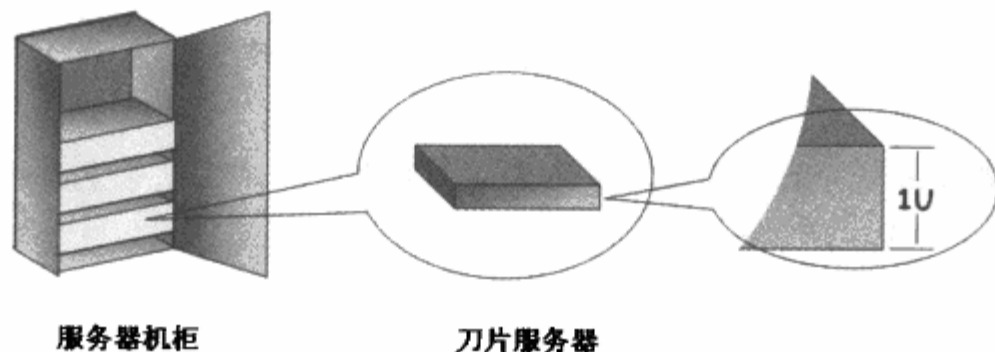


图 13-22 服务器机柜和刀片服务器

机架式服务器的单位为 U，其机箱的底面积是一定的（否则无法正常放入机柜中），U 代表的是其厚度，1U 大概相当于 4.445 厘米。机架式服务器一般为 1U 或 2U，极少能达到 4U 以上，而小型机一般都要几 U 到十几 U 左右。

同时，机房中放置的不仅只是小型机或机架式 PC 服务器这类程序运行的容器，很多时候一个大中型应用系统需要进行大容量存储，如一个视频网站的 Web 服务器就需要处理频繁的视频上传业务，这些高速海量数据显然单靠服务器的硬盘无法消化，所以磁盘阵列就是必不可少的了。

磁盘阵列的主要实现手段就是 RAID（Redundant Array of Independent Disk），即独立冗余磁盘阵列，RAID 通过将多个廉价的硬盘进行合并与控制，使得在操作系统看来 RAID 就是一块超大的硬盘。RAID 的优点就是速度快、廉价、容量大，通过合并多块硬盘，可以达到 PB 级别的数据存取。

RAID 的另外一个优点就是提供多种数据校验和冗余措施确保数据的正确性和安全性。RAID 有多个标准，比如 RAID 0 代表没有冗余，而 RAID 5 代表具有奇偶校验的分布式磁盘存储等。

“怎么样，蔡佳娃？这下对于一个软件项目从设计到开发，然后经过测试，再到部署的过程是不是有点明白了？这些过程有些你可能已经经历过，有些是你以后会经历的。”

“嗯，哈哈，这下把来龙去脉都看了一遍，至少以后吹牛不会心虚了。”

“瞧你这点出息，另外，我们谈到的很多内容都只是一些简单的介绍，要想真正掌握这些知识，平时一定要多看书，多学习，多积累啊！”

## 13.6 虚拟化与云计算

虚拟化的思想并不新鲜，相信很多读者都明白虚拟内存、虚拟机是怎么回事。本节将要探



讨的虚拟化是为企业在服务器方面的设计提供高效而绿色的解决方案，同时虚拟化技术与目前计算机行业最前沿的研究方向云计算也有着非常密切的关系。本节就来对这两种技术做一个简单的介绍。

### 13.6.1 举杯邀明月，对影成三人——虚拟化

“师兄，我们公司前两天说要搞什么服务器虚拟化，整得挺玄乎的，虚拟化是什么意思啊？”

“你应该知道虚拟内存的原理吧？”

“嗯，就是将硬盘上的空间虚拟化为内存以扩充内存的技术。”

“好，有了这个前提我再给你讲就好理解了。”

企业如果需要进行大业务量的计算，就需要购买昂贵的服务器，而有些公司的实力有限，为了达到需求，只好采用本书前面介绍过的集群技术搭建 PC 集群，这在一定程度上实现了高级服务器的效果。这本身也可以看作一种虚拟化，将多台机器虚拟成一台逻辑机器。

上述只是问题的一个方面，与那些无力购买昂贵服务器进行计算的公司或机构相比，有些公司或机构有财力配置高级的服务器，或者搭建高性能的集群，但是真实的应用中系统的负载却非常低，导致系统大部分时间都处于超低负荷。图 13-23 的漫画指出了这种服务器浪费的现象。

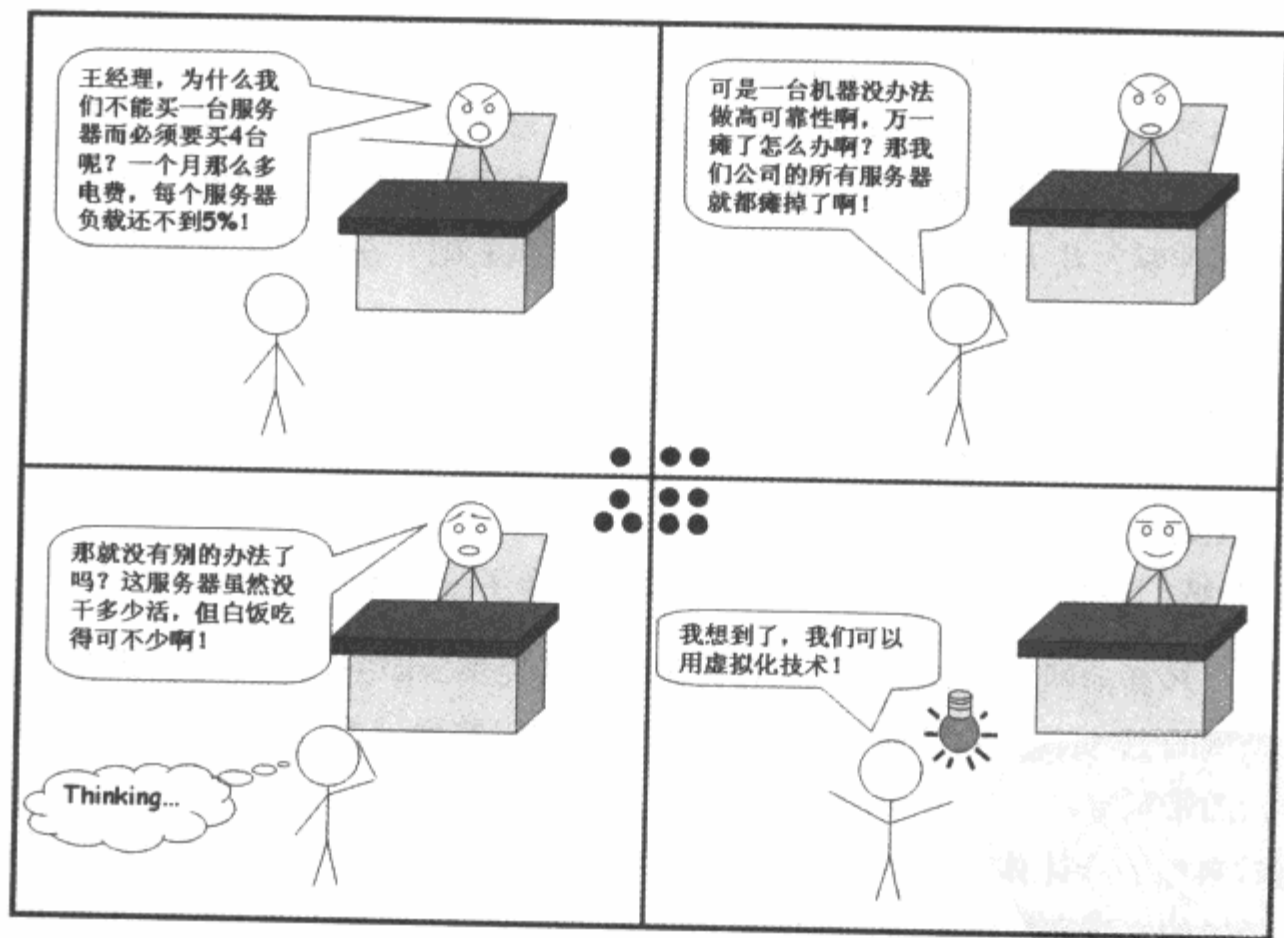


图 13-23 对系统的高利用率导致虚拟化技术

实际中，这种“朱门酒肉臭，路有冻死骨”的资源闲置现象十分常见，这些“郁郁不得志”的高性能服务器或集群不仅是对企业投资的浪费，同时每天吃掉的“白饭”如电能也一点不比满负荷工作状态少多少。这凭空消耗掉了很多资源，往大了说，增加了碳排放，着实不适合目前节能减排的大环境。这种浪费现象都可以使用虚拟化的技术加以缓解或去除。

“师兄，那这种虚拟化具体是怎么做的呢？”

“虚拟化具体实现起来是比较复杂的，但是我们可以来简要谈一谈原理。虚拟化采用硬件或软件技术，将一台高性能的服务器虚拟化为多个逻辑服务器。”

“就像分身术一样吧。”

“嗯，差不多，这些逻辑服务器互相之间是独立共处的，可以执行不同的任务，比如一个大型服务器，可以分割作为两台 Web 服务器、两台数据库服务器，这样的话一台高级服务器的价钱，可以收获 4 台服务器的效果，同时还解决了高可靠性的问题，绝对是超值啊。”

虚拟化技术可以真正地节约开支，将服务器资源的利用最大化，同时虚拟化技术还是后面将要介绍到的云计算的基础技术之一。

目前市面上提供高级服务器虚拟化技术的领跑厂商是 VMware，产品为 VMware ESX Server（注意不是 VMware Workstation 哦），同时 Sun 公司（xVM Server）和微软也都提供了一些在 Solaris 和 Windows 平台下的虚拟化解决方案。

### 13.6.2 云中谁寄锦书来——云计算

要说目前在计算机行业玩什么最前卫，那肯定是非云计算莫属了。在 IT 界如果连云计算这个名词都没听过，那可就真是 out 了。云计算不只是个新潮词汇，它还代表着未来网络的发展方向，本小节就来介绍云计算给读者朋友们认识认识。

回顾上一小节曾经提到的那个漫画（图 13-23），在漫画的最后王经理提出采用虚拟化技术将一台服务器虚拟化为多台服务器工作。这个时候王经理的老板肯定还会要问：那我们多出来的那三台服务器干嘛使啊？花大价钱买来的，总不能当古董供着吧？在读完本节的内容之后，这个困难就迎刃而解了。

“既然我们提到了虚拟化，就顺便谈谈云计算吧。”

“云计算啊？这个听过，很高妙的东西吧？我曾经随便看过一些介绍，不愧是云计算，结果把我整得云里雾里的。”

“呵呵，是吗？那我今天再来忽悠你一回，看你还会不会迷糊。”

云计算是一种新的网络计算技术，云计算为使用者提供云服务，云计算提供的服务内容几乎涵盖了所有类型的 IT 资源，除了最简单的计算服务、存储数据服务，还包含了使用云提供的应用程序、使用云的带宽等。

一个比较典型的云计算应用实例就是《纽约时报》的文档转换工作，这项工作需要将其 100 多年以来发行过的文章和图像等信息转化为 PDF 格式的文档。这项不需要很高智商但却很烦琐的工作被人认为需要至少一个半月，且需要投入大量经费购买设备。而一个工程师通过租用 Amazon 提供的一个云计算平台，只用了不到一天就完成了。这就是云计算的威力，而且只是一个小小的方面。

“师兄，那么云计算到底能干什么啊？”

“可以这么说，目前网络能提供给你的，云计算也可以提供给你。而网络不能给你的，云计算

照样能给。比如你需要做一个超复杂的科学运算，你可以将其提交到云计算平台，然后云计算平台就会调用它的资源为你运算。”

“嗯，这个的确是挺方便的。”

“你向云计算平台提交的可以是任何需求，比如要求它为你编译并部署一个应用程序等，你还可以通过云计算平台使用一些你在自己 PC 上无法使用的软硬件资源等。”

云服务能做的事远远不止故事中所提到的那些，云计算充分利用了网络中的闲置资源，为用户提供包括软件和硬件在内的各种资源，用户不需要有自己的基础设施，所有的事情都通过云计算来完成，只要提交了相应请求，就可以等着接收“云端的声音”即可，如图 13-24 所示。

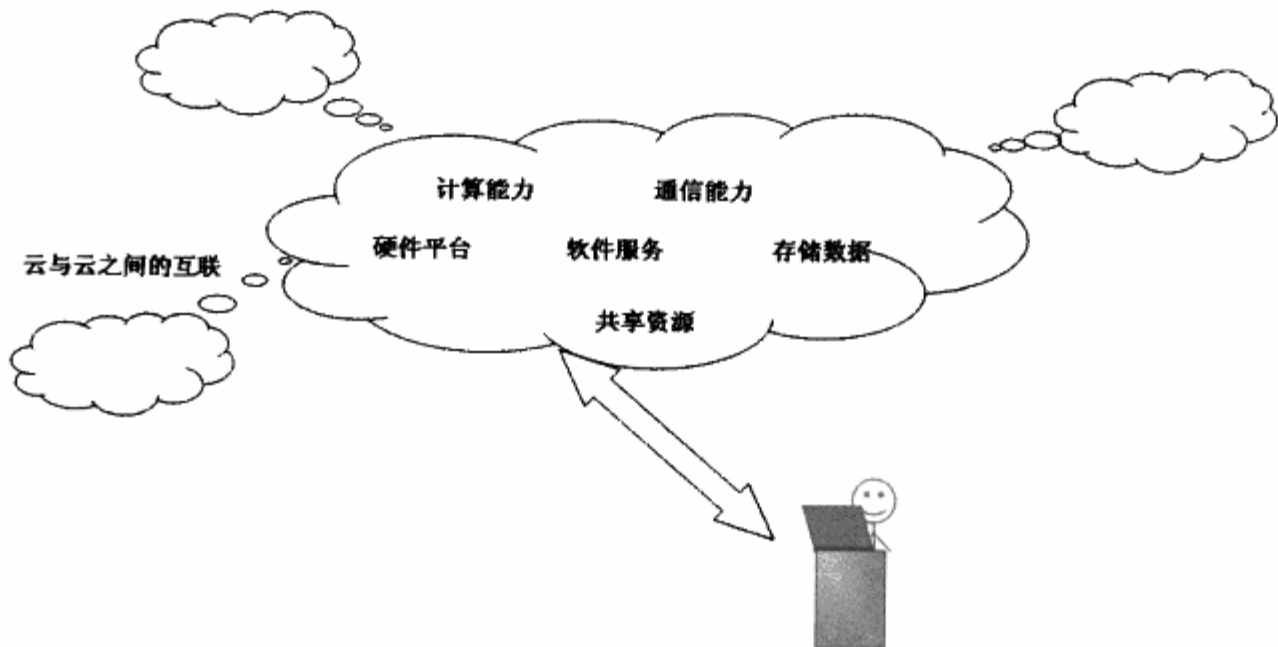


图 13-24 来自云端的声音

云计算的实现也是比较复杂的，其基础性的技术就是在前面小节中提到过的虚拟化。在云平台中，为了实现对用户提供完美的服务，需要大量使用虚拟化技术进行抽象，包括操作系统、平台、网络以及程序的虚拟化。同时云计算中还广泛涉及软件打包、机器映像等技术。

**提示** 读到这里大概读者朋友也对前面王经理所要面临的问题有了初步的解答，是的，王经理最好的办法就是再次通过虚拟化将闲置下来的服务器包装成云服务租给其他的公司使用，这样不仅充分利用了资源，还可以实现额外的盈利。

云计算主要有以下好处。

#### ● 实惠

所有的资源都可以从云平台获得，不需要购买进行工作的基础设施，如昂贵的硬件或是需要高额付费的软件等。这样大大节约了成本，原来没有财力去做的事情，都可以交给云计算平台来价廉物美地轻松完成。

#### ● 高效

云计算可以实现整合网络内部的大量资源，从而提供高效的服务。目前提供云服务的厂商有 Google、Amazon 等，这些云平台的计算能力都是非常强的。

云计算的发展将给网络带来很大的变革，云计算提供了新的商业模式、新的享受互联网的模式和新的软件开发模式。目前提供云服务的厂商越来越多，相信未来的网络世界一定是云的天下。

## 13.7 本章小结

本章介绍了一些 Java 开发人员在开发过程中会遇到的纯编程工作之外的相关知识，这些内容虽然不如开发代码富有激情，但却是一个软件系统生命周期里不可或缺的部分。因此，希望每位读者都能够了解这些知识并能够熟练运用。

本章的最后简单介绍了一下虚拟化技术和云计算，这些知识看起来离我们可能还很远，但却代表着未来的发展方向，因此了解一下还是很有必要的。与前面的章节类似，本章介绍的知识不会很深，有兴趣的读者可以自行深入学习。

# 第 14 章 杂 项

前面的 13 个章节中，本书已经把 Java 开发人员在职场江湖中生存发展所需要的方方面面的知识和道理做了一个张弛有度的介绍，相信读者已经在技术和思想以及个人态度上有了很大的提高。本书的最后一章将主要做一些拾遗补漏的工作，向读者谈谈做一名高级开发人员所需要注意培养的能力。希望读者朋友在阅读完本章之后再接再厉，向着自己的人生目标阔步前进。

## 14.1 专业英语不能不熟练

这里提英语或许不太恰当，毕竟本书是讲 Java 这门编程语言的，谈英语这种自然语言有些跑题，但是开发人员毕竟还是人，与机器的交互可以通过 Java，与人的交流就必须采用自然语言了。而英语作为目前自然语言中的“Java”，具有应用广泛、跨国度等优良特性。因此，深奥的不说，与 IT 相关的专业英语还是必须要熟练掌握的。

### 14.1.1 向高新技术看齐

“师兄，快到元旦了，今天我来请你吃饭，谢谢师兄一直以来对我不辞辛苦地教诲，把我从一个一窍不通的菜鸟变成了如今能够独当一面的开发人员，师兄你辛苦了啊！”

“呵呵，谢来谢去的，我怎么越听越像是感恩节呢？你能有这样的进步更多还是靠自己的努力，我并没有起太大的作用。说起来，我也有很多后悔没学好的东西呢！”

“啊，师兄也有遗憾哪？那是什么啊？”

“不是别的，就是英语啊，哎，要是能回到过去，我肯定把六级给过了，然后规定自己无论使用何种软件工具，一律全要英文版的，API 文档也是。”

“啊，英语真的很有必要吗？”

众所周知，IT 这个行业纯属“舶来品”，虽然我国的 IT 业在起步之后也取得了非常显著的成绩，在世界上也有了一些拿得出手的品牌，但是整个 IT 行业的中心仍然在欧美（主要是美国），最新的技术与理念也大都产自这些地方，因此从事 IT 行业的人为了技术，必须首先逾越语言这个鸿沟。

不过各位读者不要误会本书的意思，培养开发人员所需的英语能力并不需要刻意去背单词甚至听录音，英语只是扫清开发人员工作中不应该存在甚至严重桎梏发展的语言障碍。开发人员需要根据自己对英语能力的实际需求来提升自己。一般来说，工作中需要使用英文的场合有如下几种。

- 阅读文档

文档是英语这个拦路虎最爱出没的地方，开发人员在工作中可能遇到很多的文档，有的是客户的需求文档或电子邮件，有的是关于技术的帮助说明文档，有的则是来自外国的论文或帖子，这些对于英文阅读能力都是有一定要求的。



- 参加技术会议

技术会议是 IT 生活中很重要的一个部分，技术会议有一些是以汉语为主要交流语言的，但是更多的一些国际性质或规模较大的技术会议上讲的都是英文，这时候不要指望同声传译的帮助，因为能将晦涩难懂的技术术语翻译完好的同声传译实在是太少，所以在一些以英语为主的技术会议上，听不懂英语会很郁闷。

- 与国外的技术人员或客户交流

不管是身在外企还是国企，与国外的技术同行或客户交流的机会还是不少的，这种场合就必须要有英语的表达能力了。

另外，不仅是在外企，在国企内也经常听到一些老板或经理说话中夹杂着一些英文单词，这种话说起来要比纯汉语更加贴切，因为有些技术术语不容易翻译，而且这么说显得很时尚、很专业。所以如果一点英语都听不懂的话，就显得很“土”了。

### 14.1.2 等到中文版的时候

“师兄，实际上工作中不就是偶尔会遇到一些英文的文档嘛，没必要学习吧？迟早会出中文版的不是吗？”

“哦，何出此言啊？”

“我记得有人说过，当今世界，谁忽略了中国市场，谁就忽略了财富，忽略了做大做强的机会。所以不用着急，甭管是新技术还是什么东东，一定会为我们奉上中文版的吧。”

很多开发人员往往技术方面很牛，但就是专业英语的能力不够，这些人拒绝学习英语的一个原因就是：学什么英语，还不是要出中文版？这些人就如同图 14-1 漫画中的 A 君一样，太小瞧英语水平的重要性了。

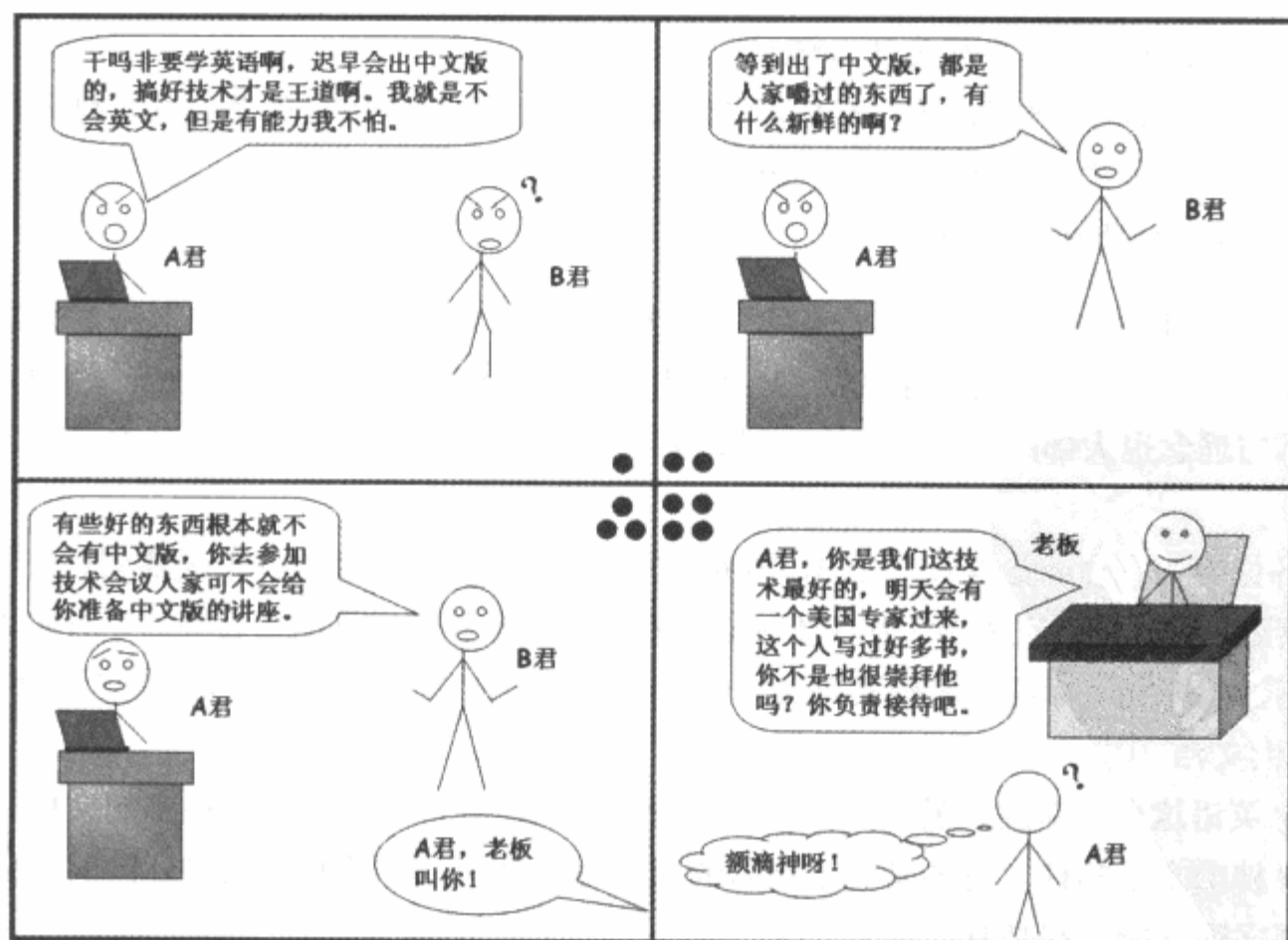


图 14-1 不重视英语的后果

前面也曾经提到过，IT行业真正的技术中心在国外，当一门新技术出现的时候，往往在很长一段时间内都只有英文版的介绍。如果不去硬着头皮研究而是坐等的话，当中文版出来之时，或许这门曾经新奇的技术早已经有了成熟稳定的版本，会的人已经如山花烂漫般地漫山遍野了。

只依靠中文资料除了无法走在技术领域的前列以快取胜之外，还有以下两个不良后果。

- 很快达到知识干涸

由于过分依赖中文资料，在对知识进行深入探索的时候，会因为中文资料的有限而很快走到尽头，因为一种技术的高级应用的知识往往都是英文，除非这门技术出自国人。

- 无法展现自己

IT不是个学校，不需要一味地去聆听和跟进别人，有的时候需要将自己某个领域独特的见解或成果展示给别人，如果专业英语的能力不够强的话，也会损失很大一批观众。

因此在软件开发这条路上，是要做个探路者，还是要做个追随者，决定权完全在自己。是否应该提高自己的专业英语水平，也完全因人而异。

### 14.1.3 做一个大牛的需要


“总体来说，良好的专业英语能力，也是做一个技术大牛的需要。”

“技术大牛也很需要英语能力吗？”

“对啊，在我所认识的高人中，几乎没有英文水平很差的。因为一般一旦在英语这个非常次要问题上暴露了不足，那么也就很难继续深究其他高深的问题了，更别谈做什么大牛了。”

想要做一个技术大牛，注定走的道路不能和普通的开发人员一样，要想真正地提升自我，对于专业英语这个辅助工具就必须有非常好的驾驭能力，绝对不可以出现被英语所羁绊的状况。

真正的技术大牛，对于新技术不能坐等，要有先睹为快的勇气和能力。而在对一些问题进行深入研究时，也必须能够无障碍地获取英文版资料的精华。同时，技术大牛在与别人进行技术交流时，也不可能因为语言的鸿沟而出现面面相觑的情况等。

 **提示** 在此重申一下，对于专业英语的学习，够用即可，不可将精力过分花费于此，技术思想和辅助工具之间的区别，还是要分清的。

## 14.2 维护大脑这个数据库

一个人的智商，可以看做是计算机的CPU运算能力，它决定了能够转多快。但并不一定智商高的电脑就更好用，硬盘中所安装的软件也是一个非常重要的衡量优劣的标准。一个人的技术储备就相当于计算机的硬盘，当一个程序运行的时候，有时需要访问硬盘获取数据，如果硬盘上的数据不能够满足需求，就相当于一个人在工作中遇到了无法解决的问题。

开发人员在工作中，查找资料是一个很常见的工作。在上面的比喻中，查询自己的知识储备算是一种查找资料的方法，那么如何形成这个知识储备库呢？同时还有哪些知识的数据库可以自由访问呢？本节就来简单谈一下维护知识储备库的问题。

### 14.2.1 书到用时方恨少

技术书籍，算是每个开发人员知识储备数据库中一个比较重要的表，而维护这张表需要投入

的精力也不小，但是收到的效益也是非常大的。

“蔡佳娃，从学校走出来，你带出来的书有什么啊？”

“啊？什么书？我离校之前把书全都卖掉了，带出来的是卖书钱。”

“那你现在也应该有一个小规模‘藏经阁’了吧？”

“比起师兄你的大书架来说，我的那几本书就可以忽略不计了吧。”

“呵呵，看来你很不注意扩充自己的知识储备啊。”

书的第一个作用就是作为教科书来指导和辅助学习，这是对于书的一个基本要求，如在学校里面学习各种课程时就属于这种用书之道。同时自己购买书籍并对其进行深入学习也属于这种使用方式，这当然算是一种知识储备的方式，但还不是本节将要介绍的。

在开发人员的世界里，书的另外一种用法就是将书买回来以后，并不对其进行细致入微的学习，而是大致了解整本书的写作背景和主要内容，从中提取出一些有用信息即可结束对这本书的探索，将其归入到“藏经阁”了。

“师兄，不会你的那个大书架上的书你基本上都没有读完吧？你这样买了却不认真看，不觉得很浪费吗？”

“正解，我要是把那些买过的书全部读完，那还不读到猴年马月，时不我待啊！”

“那你这么‘不求甚解’，身为菜鸟的师弟，我可要提出一些批评啊。”

“我对于书的使用方式不能算是不求甚解，谁都知道做一名软件开发人员有多累多忙，如果把买了的书全部精读一遍的话，时间上根本不允许嘛。”

在《资治通鉴》中有一篇讲的是孙权劝谏吕蒙读书的故事，原文是：“但当涉猎，见往事耳”，意思就是对于书籍只需要粗略地阅读，了解一些历史上的事件就行。对于技术书籍也是这样，买回来之后大可以“浅尝辄止”，不过这个“浅尝”可以浅，但必须对整本书有一个整体的了解，这样知识储备数据库中就可以添加进这条记录了。

在开发工作中，遇到某个实际问题时，如果自己“涉猎”过的书籍多，应该很快地在数据库中找到可以解决这类问题的书籍，然后进行深入研究，最后使用新学到的知识完满地解决问题。这种知识储备的方式非常高效。

书到用时方恨少，千万不要用时找，所以为了让自己在解决问题的时候不至于捉襟见肘、毫无头绪，就应该学会把工夫下在平时，不断地为自己的知识储备库插入新的记录。这样在遇到棘手问题时就算不能立刻求解，也会大致明白需要到哪本书中去寻求答案。

### 14.2.2 让积累成为一种习惯

“师兄，我明白了，原来买书也是一种技术储备啊，这样等到需要的时候直接就能查找得到，就像是数据库中建立的索引一样，快速高效。”

“是啊，除了买书这种储备方式，还有很多其他的积累方式。还记得我们很早就提高到过的‘小仓库’问题吧，不知道你最近有没有更新你的小仓库啊？”

“啊，好像是没有诶。哎，总是会忘记将重要的东西记录下来。”

积累是一种增加知识储备的有效途径，收藏技术书籍从某种意义上来说也算是一种积累，本

书在第 2 章曾经提到过的维护一个电子小仓库也是在做积累。收藏书籍这种积累比较容易实现，小仓库维护起来就有些困难了。

在工作中有时候一些本该积累的东西经常会被忽略，等再次需要用到的时候，由于没有索引，只好在浩瀚的记忆库中去搜寻。这倒不只是浪费时间的问题，而是多半情况下被忽略的东西早已经成为垃圾被大脑的垃圾收集器给回收掉了。

举个例子来说，在搭建一个 Tomcat 集群的时候通过多方资料查找和反复的实验，终于搭建成功，这时候编写一份详细的工作手册来记录整个过程对于未来的再次使用就非常有必要了。好多东西被忽略的原因都是开发人员高估了自己的记忆力，所以一定要努力让积累成为一种习惯。

“师兄，看来除了花大价钱收藏技术书籍之外，我还要自己主动地去积累一些工作中宝贵的东西呀，我荒废已久的小仓库，还是得重新开张啊。”

“是啊，尽信书则不如无书，虽然书中自有黄金屋，很多时候还是需要靠自己千淘万漉，吹尽黄沙才能看到金子的，所以积累绝对不是个简单的工作。”

当积累成为一种习惯，一个人的知识储备库就会慢慢增加。要明确的一点是所有的积累都是为了将来查找的方便和快速。积累是最懂得感恩图报的，下面的漫画（图 14-2）就说明了这一点。

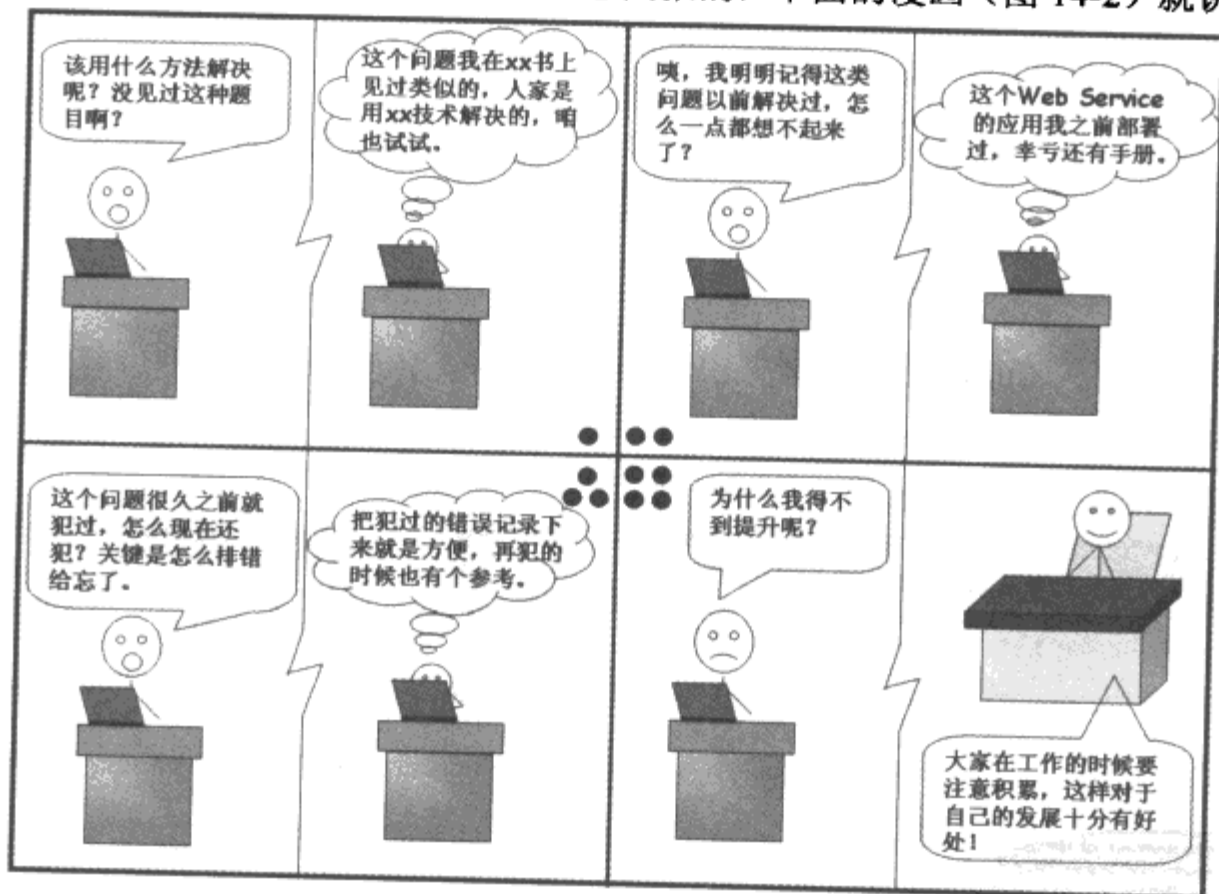


图 14-2 积累与否的结果对比

### 14.2.3 搜索引擎的使用

前面也提到了，积累的目的就是为了将来需要用到的时候方便查找，这里的查找是查找自己大脑中的数据库。不过，外面的世界中还有一个更大的数据库可以去检索，那就是互联网。而要想从互联网上查找到有用的信息，最简单实用的方法就是使用搜索引擎。

“蔡佳娃，一直跟你说要注意积累，增加自己的知识储备。其实在实际工作中，开发人员搜索较多的，除了自己的脑袋，就是互联网了。”

“嗯，就是搜索资料吧，这个谁不会啊！”

“看看，又孤陋寡闻了吧。你那种搜索资料的方式实在是太小儿科了，搜索资料的最高境界可是讲究搜索技巧的哦。”

在互联网上查找资料，并不仅仅是个力气活，也是个技术活。在搜索栏中输入关键词按回车键这个工作谁都会，但是真正的高手是采用搜索技巧来快速搜索资料的。而学习搜索技巧的好处，就是可以将万千的搜索结果进行主动的筛选过滤，只将最有用的有限结果罗列出来。

相信大家应该和笔者一样，在进行技术知识方面的搜索时，使用的都是 Google。选择 Google 的原因很多，最重要的在于 Google 是目前最好的英文搜索引擎，作为一名开发人员，英文资料的重要性大概不必重申了。

下面就来介绍几种开发人员在搜索资料时比较实用的 Google 搜索技巧。

### 1. 逻辑操作符的使用

在 Google 中搜索所输入的关键词是可以有逻辑关系的，默认情况下，所输入的用空格隔开的关键词组是“与”的关系，如输入“MySQL 下载”表示搜索同时包含“MySQL”和“下载”的网页。而如果需要表示多个关键词之间的“或”关系，就需要使用“OR”操作符，具体语法为：

1 关键词 1 OR 关键词 2 OR 关键词 3……

如搜索“Struts OR Spring”将会返回所有包含“Struts”或“Spring”的网页。如果需要在搜索结果中去掉会出现某种关键词的情况，就需要使用“非”操作符，其语法为：

1 关键词 1...关键词 n -屏蔽关键词

假设需要搜索有关“JavaFX”的网页，但是不希望搜索结果中出现 Sun 的官方字眼，就可以在搜索框中输入“JavaFX -sun”以屏蔽掉包含关键字“Sun”的网页。不使用屏蔽关键词和使用屏蔽关键词的搜索结果如图 14-3 和图 14-4 所示。

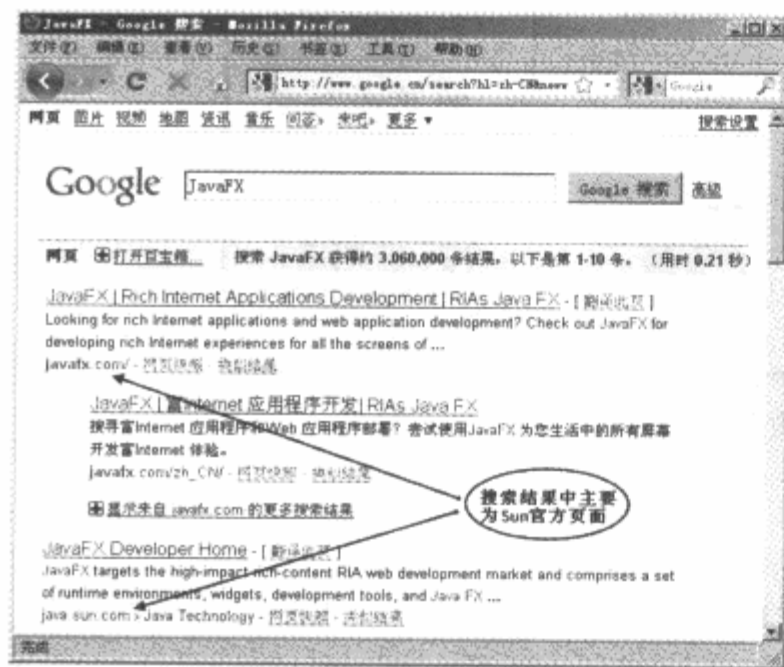


图 14-3 不使用屏蔽关键字搜索结果

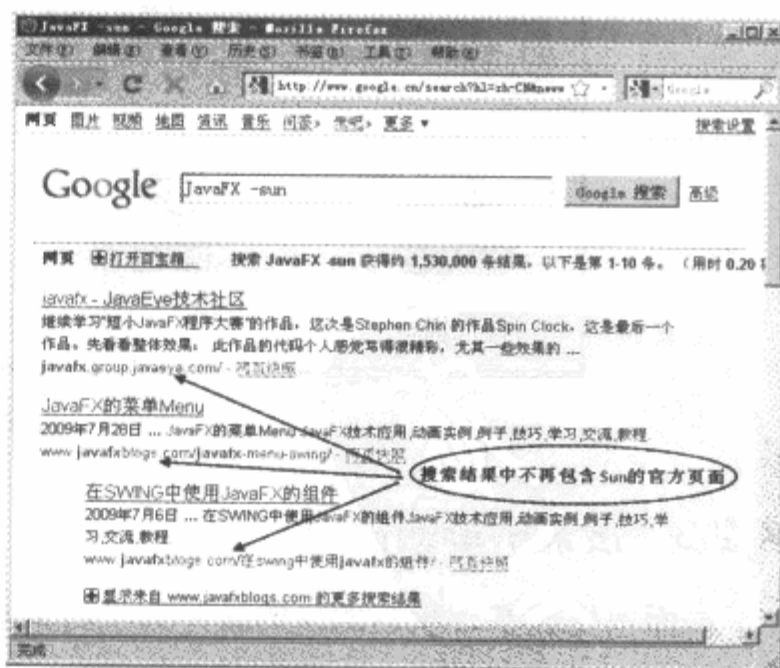


图 14-4 使用屏蔽关键字搜索结果

**提示** Google 不区分大小写，所以在搜索框中输入“Sun”和“sun”其结果是一样的，但是对于本节涉及的逻辑操作符如“OR”则必须为大写字母。



## 2. 强制整体搜索

输入一个较长的关键词时，Google 会聪明地将其拆分为几个部分，然后对这几个部分采用逻辑“与”关系进行搜索。但是有些时候，需要对关键词整体进行匹配而不希望搜索引擎将其拆分，这时就可以使用强制匹配搜索，其语法如下：

- 1 “不希望拆分的关键字”

将不希望拆分的关键词用“”包裹起来，如搜索“Java SE 6.0 编程指南”时可以用双引号括起来告诉 Google 要强制整体搜索，采用整体搜索的结果如图 14-5 所示。

## 3. 通配符

在 Windows 中搜索文件或查找文件可以使用通配符，使用 Google 搜索资料也可以使用通配符，不过 Google 支持的通配符只有“\*”。代表一个字符，而且包含通配符的关键字必须用引号“”引起来，例如在搜索书名时输入“Java 教\*程”。由于“\*”代表一个字符，而网络上“教”与“程”之间是没有汉字的，搜索结果应该为空，其结果如图 14-6 所示。

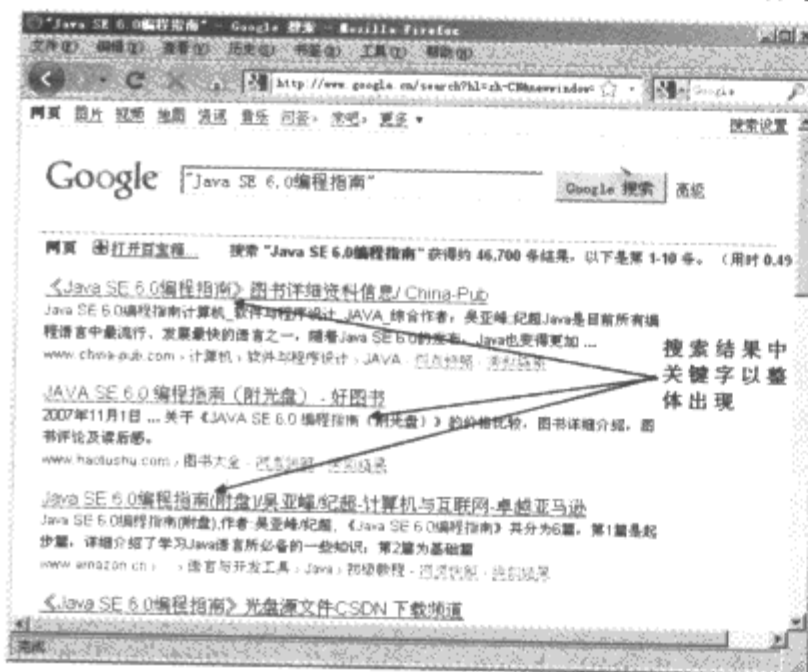


图 14-5 强制匹配搜索结果

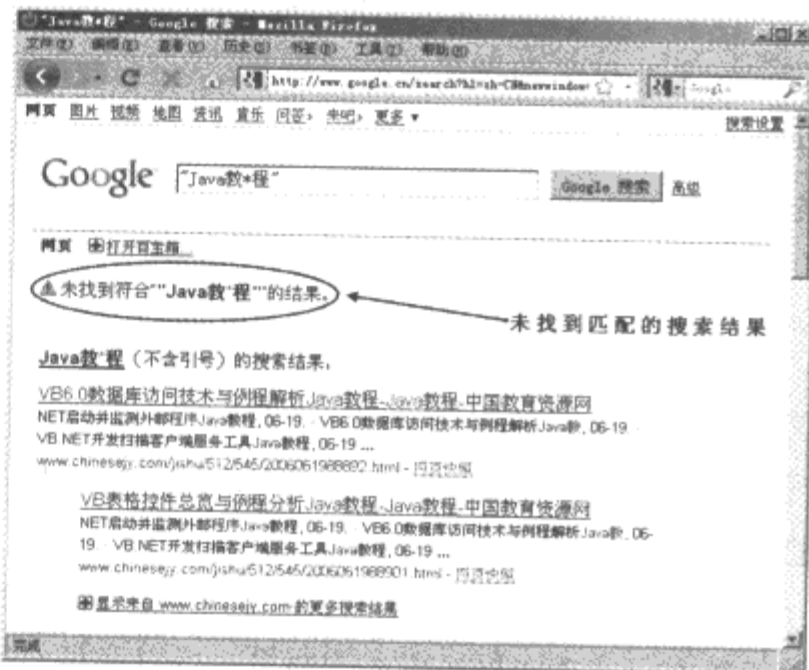


图 14-6 采用通配符搜索结果

## 4. 强制搜索忽略字符

很多时候用户输入的某些词是不影响搜索结果的，如“的”、“了”、“呢”等，英文中如“www”、“com”等，Google 会在搜索时忽略掉这些高频字符以提高搜索效率。但是有些时候必须要将这些被忽略的词包含进来，这时就可以使用强制搜索操作符“+”。

强制搜索符的语法为在需要强制搜索的易忽略关键词前面加上“+”，例如要搜书籍《我的编程感悟》，为了使结果更精确，可以在搜索框中输入“我+的编程感悟”，其搜索结果如图 14-7 所示。

## 5. 指明文件类型

Google 不仅支持对网页的搜索，还支持对指定文件类型的搜索，如 Office 套件中的.doc、.xls、.ppt 等文件以及互联网上人气最高的 PDF 文档等其他许多常用的文档。采用指明文件类型的搜索方式可以精确地搜索一些需要的文档资料。

如要搜索关于云计算的 PDF 论文资料，可以在搜索框中输入“云计算 filetype:pdf”，搜索结果如图 14-8 所示，可见 Google 已将所有包含云计算内容的 PDF 文档罗列出来。

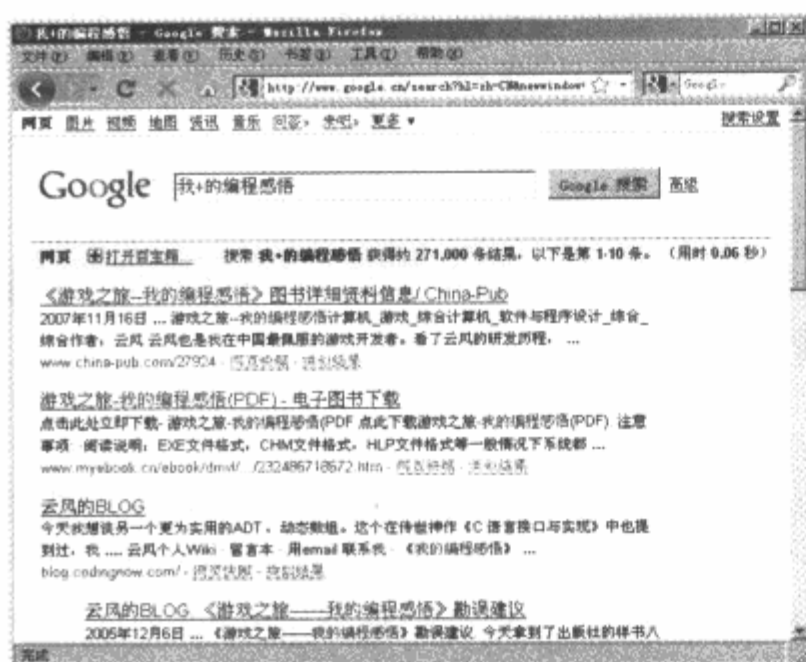


图 14-7 使用强制搜索结果

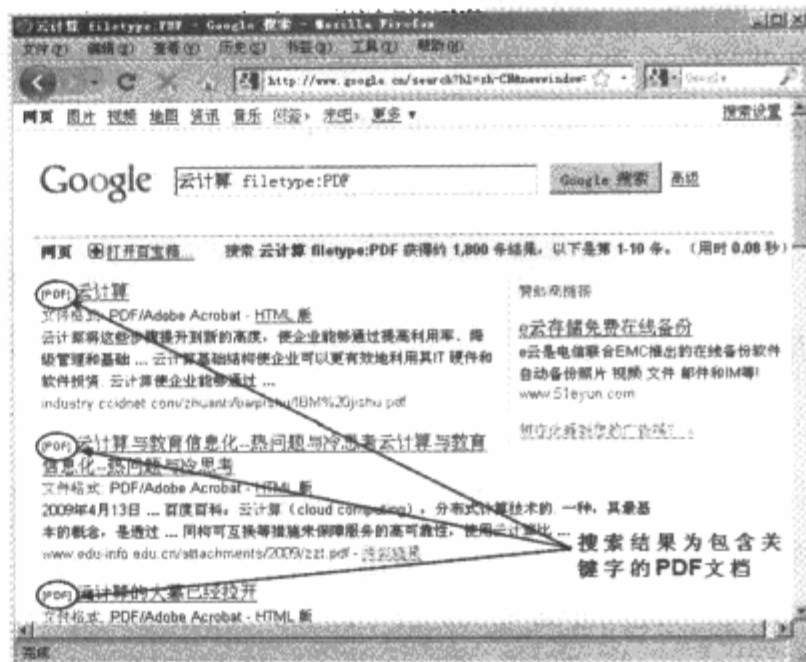


图 14-8 搜索指定文件类型示例

同时在指定需要的文件类型时，还可以用“OR”将多个文件类型连接起来，表示搜索多个文件类型中的任意一种。如输入“云计算 filetype:pdf OR doc OR ppt”，则搜索引擎会将涉及云计算内容的 PDF 文档和 Word 文档以及 PowerPoint 文档一并列出。

## 6. intitle 的用法

如果在搜索信息的时候只想看到网页标题中包含指定关键字的网页，就可以使用“intitle”语法来完成。使用“intitle”语法的格式为“intitle:关键字 1 [关键字 2]”，搜索引擎在处理这类搜索时只将网页标题（即<title></title>标记之间的信息）包含关键字 1 以及网页内容中包含关键字 2 的网页列出。

由于网页的主要内容一般浓缩在网页标题中，所以这种搜索方式可以有效地缩小搜索范围。例如搜索包含 Mashup 技术的网页，可以在搜索栏中输入“intitle:Mashup 混搭”，其搜索结果如图 14-9 所示。



图 14-9 使用 intitle 语法搜索信息

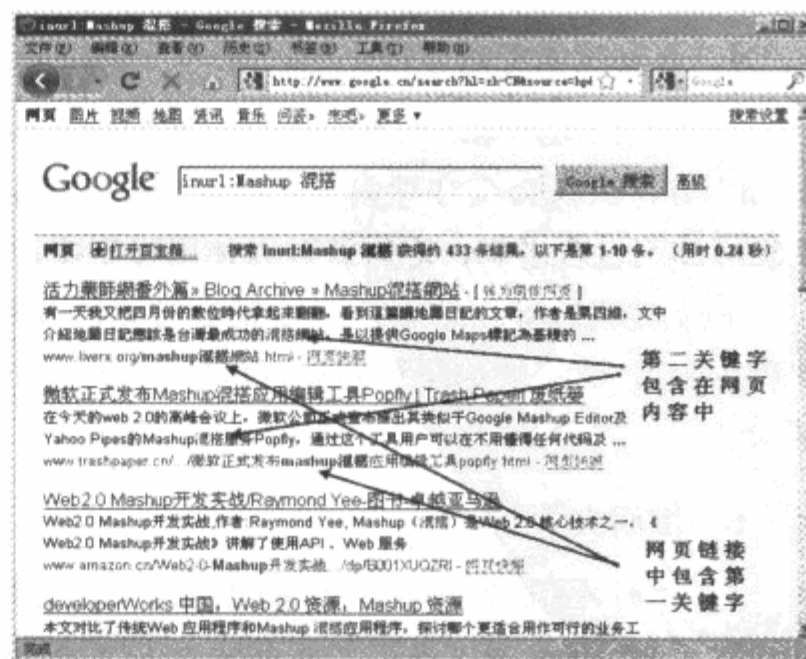


图 14-10 使用 inurl 语法搜索信息

## 7. inurl 的用法

inurl 语法和 intitle 语法类似,只不过 inurl 语法关注的不是网页标题,而是网页中的超链接。其格式为“inurl:关键字 1 [关键字 2]”,表示搜索网页链接中包含关键字 1 以及网页文档中包含关键字 2 的网页。如要搜索含有“Mashup”链接且与“混搭”技术相关的网页,则在搜索栏中输入“inurl:Mashup 混搭”,其搜索结果如图 14-10 所示。

**提示** 对应于 intitle 和 inurl,有 allintitle 和 allinurl 语法与之对应,其含义是搜索的网页标题或链接中必须全部包含所给的关键字序列。

## 8. site 的用法

有的时候希望所搜索的信息来自于同一个网站,就可以使用 site 语法,site 的语法格式为:“关键字 site:网站域名”。如要搜索来自 CSDN 网站的有关手机游戏开发的网页,可在搜索栏中输入“手机游戏开发 site:www.csdn.net”,其搜索结果如图 14-11 所示。

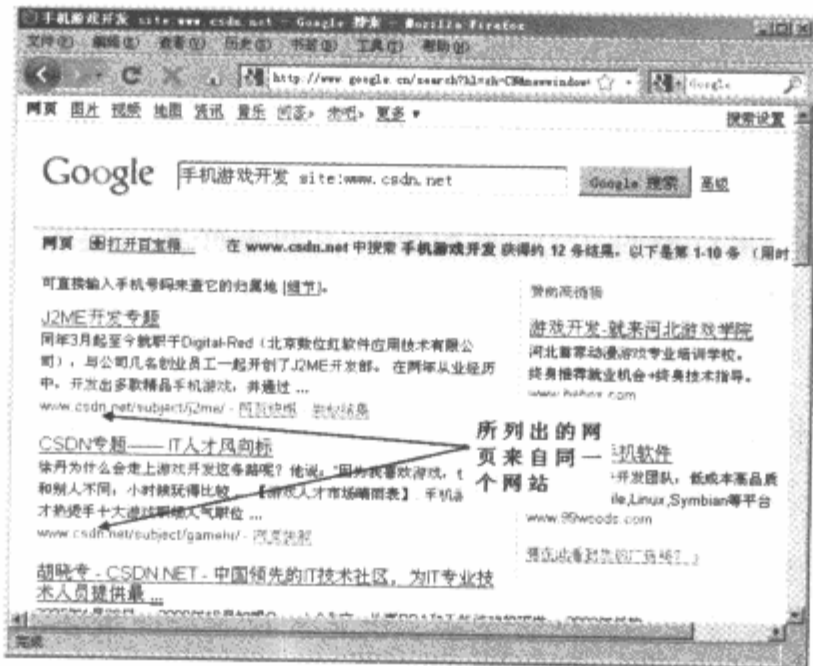


图 14-11 使用 site 语法搜索信息示例

需要注意的是在使用 site 语法时,网站域名的前端不可以有“http://”,域名后端也不可以有诸如“/product”之类的子目录。

“怎么样,蔡佳娃,这些搜索技巧你以前没用过吧?”

“显然没用过嘛,看来搜索东西也是一门学问呢。”

“那是当然,我们所提到的这些搜索技巧,都是为了同一个目的:缩小结果范围,提高命中率。这才算是真正发挥了搜索技巧的优势。”

本节介绍了几种更新开发人员知识储备库的方法,其中提到了收藏技术书籍、主动有意识地去积累以及如何更好地利用搜索引擎。这些都像是在大脑中的知识数据库中插入新记录,这样才能保证日后需要的时候能够有据可查,而不会返回“该页无法显示”之类的尴尬。

## 14.3 IT 人也要不务正业

IT 人首先必须要敬业,否则如何砍下一个个的软件项目,如何熬过一个个难关?不过光靠着

热火朝天的敬业精神还是不够的，必要的不务正业也会对工作有很大的帮助。本节就来谈谈 IT 人如何在工作中不务正业。

### 14.3.1 不懂数学岂不是很糟糕

数学或许是很多人在学生时代都为之头疼过的学科，但数学却是最能体现一个人逻辑思维水平的学科。如果哪位有志于投奔 IT 行业的读者朋友对于数学的学习叫苦不迭的话，那可真得引起注意了，因为数学的素养对于一名开发人员的未来发展影响很大。

“蔡佳娃，我们讨论了这么多 Java 开发技术方面的问题，现在我们也来提一些题外话。适当地了解这些题外话的相关知识，对于你以后的开发工作也是很有裨益的。”

“哦，是嘛？那是什么题外话啊？”

“首先呢，我们来谈谈数学的问题。”

“数学啊，我们会用到很多数学知识吗？不就是一些简单的循环计数之类的吗？”

“你这么想可就大错特错了，真实工作中用到的数学知识可远不止这些呢！”

为了让读者朋友们更好地理解数学在软件开发中的重要性，下面就来列举一个实际开发中遇到的真实问题，该案例描述如下。

在一款计算修筑公路预算的软件中，需要根据一定的条件求出建造公路所需的原材料如沙子、水泥等的配比情况，将该问题抽象为数学模型后为一个多元一次方程组，其中未知数的个数为 15 个，方程的个数为 16 个。

如果是 15 个未知数和 15 个方程，这个问题就很好解决。但是偏偏这个方程组中方程的个数多了一个，这个多出来的方程使得求解过程陷入了僵局。

“师兄，这个案例是你曾经遇到过的吗？那你们是怎么解决的呢？”

“是啊，当时我们谁也不懂该如何进行下去，倒是我提了一下要不找学数学的人瞧瞧看，后来就请来了一个原来学数学的员工，不过他也没有说出个所以然来。”

“哎，估计如果他学数学学得很好的话，肯定不会来这里做开发的。”

“呵呵，不管怎么说，问题还是要解决，后来项目经理发话了，每个人回去什么也不干，就去翻书，查找有关这个方程组的所有信息。结果终于找到了这个方程组的名字——超定方程组。”

超定方程组早在 17 世纪就有人做过了很深入的研究，一般情况下，由于加以未知数之上的限制条件过多，超定方程组的是没有绝对解的，只能尽最大可能在满足条件的情况下求出近似解。计算超定方程组的最小二乘解就是一种比较好的求近似解的方法。

在本案例中，需要求解的空间为公路预算中各个原材料的含量，而且方程右边的数值在实际工程中也是不需要绝对相等的，只要处于一定的范围之内即可。而最小二乘法提供的也是一个最接近绝对解的解空间。有了前人智慧的强大支撑，这个有关公路预算的问题算是迎刃而解了。

“怎么样，这种数学问题是不是只通过写代码没办法完成啊？”

“厉害倒是厉害，不过这种问题直接穷举应该也可以吧，虽然效率慢些，但至少我可以做出来啊，效率问题可以以后再谈嘛。”

“看，你还是嘴硬，好，我就再来举一个例子，看你能不能给我穷举出来。”



有些问题如果不引入数学知识，光靠计算机的主频飞转是很难算出来的，比如需要在几个村子之间实现“村村通”工程。使得每两个村子之间可以不直接连通，但必须有公路可达。每个村子之间的距离和修路的成本都不一样，要求计算出最省钱的修路方式，如图 14-12 所示。

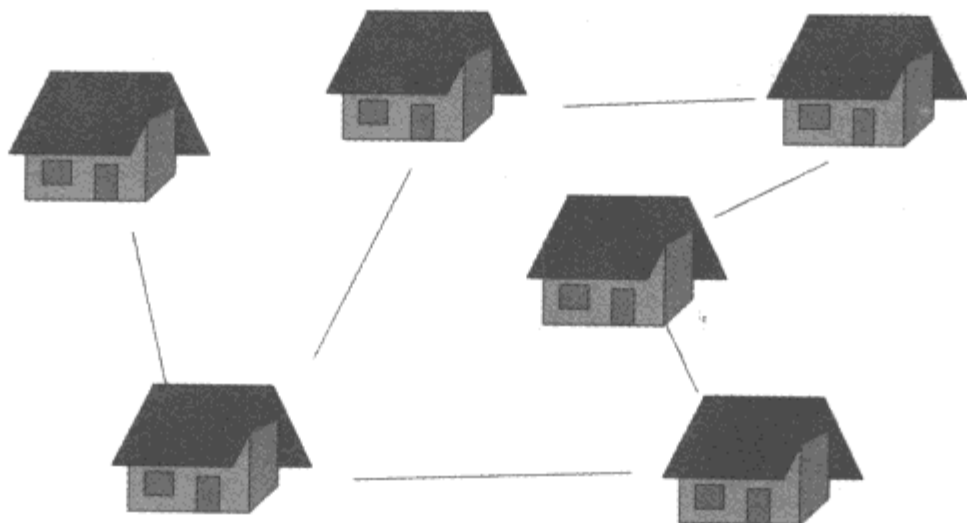


图 14-12 农村之间修公路问题

这类问题算是比较面向实际了，这样的问题该如何抽象为计算机模型呢？直接入手的话将会是个非常艰苦的过程，而且并不能保证得出结果。但是如果对于数学中图论的知识有所了解的话，这将是一个非常简单的求解图的最小生成树的问题。而且数学界的前辈早已对这个问题提出了圆满的解决方案，如最有名的克鲁斯卡尔算法（Kruskal）和普利姆（Prim）算法。

像这种初看之下很难入手的问题在实际生活中可以说比比皆是，实际生活要面对的难题绝不可能那么容易就能转变为一行一行的代码。而数学却是专门负责对现实世界的问题进行抽象和推理的学科，数学也是一门语言，一门比 Java 还要高深好多的语言。数学王子高斯曾说过：“数学是科学之王”。作为一名开发人员，笔者感受到的数学语言的强大之处主要体现在如下几个方面。

### 1. 简洁、确定

用数学语言来表示这个世界是最简洁不过的了，同时数学还是讲究确定性的学科，比如“ $1+1=2$ ”就是一个永远成立的公理（十进制下）。数学中这种简洁确定的特性用在说一不二的计算机编程中就非常适合，所以软件开发有时就像是用数学语言来解决问题。

### 2. 哪里都不能少

正当人们感慨 IT 技术越来越渗入到人们生活中方方面面的时候，孰不知数学早已在很早之前就研究领域扩展到了这些地方。大数学家华罗庚曾说过：“宇宙之大，粒子之微，火箭之速，化工之巧，地球之变，生物之谜，日用之繁，无处不用数学。”

数学涉及的学科众多，几乎包含了现实中的所有门类。所以，作为同样立志于解决各方面复杂问题的 IT 行业，当然就有必要仰仗数学这个老前辈，依托于数学这个强大的基础，创造出更加完美的解决方案来。

举个例子来说，现在医院中经常使用到的 CT 检查，其基本思想却是来自于奥地利数学家雷唐在 1917 年发表的论文，可惜他当时的思想不受世人关注。直到 50 年后随着计算机技术的发展，他的思想才被人重视起来，并结合计算机技术创造了 X 射线断层照相的技术，大大造福了人类。



## 2. 过人之处

数学最厉害的武器就是抽象，数学能够很好地将现实世界的问题抽象为数学模型。同时数学还能构造出一些现实世界中无法存在的模型，如  $N$  维空间等。数学还是进行逻辑推理的最好途径，数学虽然深奥，但却有无限的创造能力。数学甚至还可以预测未来将要发生的事，比如下一次哈雷彗星经过地球的时间等。

多了解一些数学方面的知识确实有助于开发，还是像本书前面讲的那样，对于数学的研究不必“治经为博士”，只需要知道自己目前面临的问题能否在数学领域找到符合条件的模型即可，天才的数学家们早已经为我们准备好了完备的答案。

### 14.3.2 谁说物理是白学了

如果说数学在开发人员中还多少有些人气的话，那么物理可真算是从头到尾地不招人待见了。可是这两门让不少人头疼的学科，却是对于软件开发大有裨益的。数学的尊贵之处前面已经介绍过，本小节来谈谈物理知识对于 Java 开发人员的必要性。

“蔡佳娃，数理不分家，既然谈到了数学，也就不得不提物理了。”

“啊？我与物理有深仇大恨，当年险些因为大学物理而毕不了业呢。”

“哦，既然你与物理不合，那你以后能从事的领域又会少一些了。”

“嗯？物理知识在我们软件开发行业中的用处也很多吗？”

“那当然，IT 行业可是个触手极广的大章鱼呢，怎么会少了物理这块大餐啊。”

物理是一门研究大自然的学科，是人们尝试用自己的方式来解释这个世界方方面面的途径。物理研究的范围同数学一样广大，既有宏观世界中的黑洞和宇宙大爆炸理论，又有微观世界中的夸克粒子。而在 IT 行业，物理学的应用主要体现在如下几个方面。

- 模拟实验和仿真

物理学中肯定少不了永无休止的实验，正是这些实验奠定了每个物理学理论的基础。但是有些物理实验很难在现实世界中去做，或者代价太高。打个比方说研究原子弹爆炸产生的射线扩散，肯定不能动不动就去引爆一颗然后去测量。

这时候就可以基于物理学的理论知识，通过计算机来模拟这些实验，这样既节约了成本，又提供了一个新的实验平台。同时物理还可以对现实进行仿真，如一些汽车、飞机的驾驶模拟系统等。所有的这些都可以用软件来实现，但是对于开发人员物理知识水平的要求就必须十分高了。

- 游戏开发

如果说模拟实验和仿真现实对于开发人员过于遥远和枯燥，那么游戏这个话题大概不会有人不感兴趣了。游戏中绚丽逼真的场景，就是通过物理学的知识来建立模型的。那些诸如水波扩散、飞行轨迹、爆炸喷发、碰撞旋转、风随影动之类的来自现实世界的视觉效果都是通过对物理学中的相应公式进行计算而渲染得来的。

现在市面上的游戏厂商都已经开始采用物理引擎来对场景进行更好的渲染，如一些显卡的制造商也开始在显卡产品中集成物理引擎的优化芯片。所以如果从事游戏的开发工作，具备最基础的物理知识还是十分必要的。

### ● 计算机辅助设计

辅助设计是计算机技术主要的应用领域之一，在实际中很多的辅助设计软件都是面向和物理有关的工程领域的。为了使这些软件的使用者能够一心一意地进行设计，就必须在开发过程中为其注入很多物理学上的知识，如进行汽车或建筑设计的时候就需要考虑到结构力学、流体力学等方面的知识。

“蔡佳娃，看出来了吧，从中学到大学，物理可不是白学的哟。”

“呵呵，师兄，我记得很多年前有句口号：‘学好数理化，走遍天下都不怕’，看来这句口号没有过时嘛。”

“的确啊，这些物理知识本来就是从客观世界得出来的，我们的工作就是将这些理论编写进我们的程序中，并以此建立计算机的虚拟世界。”

“听起来蛮壮观的，了解一定的物理知识实在是不会亏待自己的啊。”

### 14.3.3 一起来不务正业吧


前面介绍了数学和物理知识在软件开发中的重要性，现在来对其做一个小结。与冯诺依曼和图灵这样的大家相比，各位读者包括笔者在内都只能自叹不如，而科学史上还有着数不清的天才，所以应该明白这样一个浅显的道理：我们所能想到的、碰到的问题，不出意外早已经有人想到了，而且做出了精湛的解答。

因此作为普通人的我们，根本不需要对其做十分深入的研究。能够站在巨人的肩膀上运用好这些理论，就已经是大牛了。而要运用这些理论，就必须要对其有一个哪怕是最浅显的掌握，至少能够做到在遇到某个问题时知道应该去哪里寻求解答。

计算机适合去解决问题，但不适合去思考问题，而数学和物理这些学科却是长于思考，所以应该让软件开发和这些理论知识结合起来才能做到最好。

“师兄，既然数理知识也这么重要，那到了我们这种程度，该如何去补课呢？”

“呵呵，数理知识只是需要浏览涉猎，你可以读一些通俗的文章或新闻介绍，了解一下当前的行情即可。我可以向你推荐一本好书《20世纪数学经纬》，读一读还是很有好处的。”

 **提示** 有兴趣的读者也可以将一些数学家、物理学家的传记作为枕边书，这样在愉悦身心的同时也大大开阔了眼界。

## 14.4 读学术论文

看到这个标题或许已经有读者退缩了，还或许有读者朋友会说：这是不是书的末尾沽名钓誉的一个唱高调啊？是的，学术论文这个词汇出现在此类书籍中实在突兀，笔者也从来没见过哪本技术书籍会将学术论文搬上书页的，作为第一个吃螃蟹的人，本书有信心通过“读学术论文”这一节为本章画一个圆满的句号。

### 14.4.1 别怕我，我是好人

之所以有些人看到“读学术论文”这个题目就会退缩，是因为人们对于学术论文的了解实在

太少，以至于无法在心中为其做一个正确的定位。本小节就来改变一下大家对于学术论文不客观的看法，将学术论文的本来还原出来。

“蔡佳娃，作为一个过来人，我有必要提醒你一下，在今后的工作中，阅读学术论文也是一个非常实用的解决问题的途径，你要在……”

“哎，停，师兄你没搞错吧？学术论文？你确定你说的是学术论文？”

“当然，我确定一定以及肯定说的是学术论文。”

“可是，师兄你也许比较厉害，我看我还是算了吧。学术论文可不是我们这辈人能玩得起的。那都是留着大胡子的学者们每天忙碌的事情呢。”

“哦？看不出来，你和学术论文之间还有些误会呢。幸亏被我发现得早啊。”

或许是受到了当年爱因斯坦发表的鲜有人懂的广义相对论论文的影响吧，学术论文在很多人的眼中都有些高不可攀，或许能让一般人看懂的学术论文本身就不算是好的学术论文。很多人认为学术论文是用来筛选强者、打击弱者的，这些片面的看法一般集中于以下两个方面。

- 一个字，难。学术论文集中了太多的理论知识、太深奥的公式或原理，并非整天与代码打交道的开发人员所能消化得了的。
- 用不上。学术论文和实际开发二者术业有专攻，互相之间没有关系，虽然是理论指导实践，但是实践还得去验证理论，索性实践指导实践更加实在。

对于学术论文的这些误会，实在是太过冤枉了。学术论文远远没有一般人想象的那么高深，至少 IT 行业的论文不是。本着实用的原则，学术论文面向的群体还是广大的开发人员，当然了，每个人的理解能力不同，看不懂的现象还是有的。

“师兄，既然学术论文不像传说中的那么可怕，那学术论文的实用价值怎样呢？”

“如果说因为人的水平有高低之分，学术论文的可读性有待考究的话，那么优秀的学术论文的实用价值则是毋庸置疑的。”

“是吗？不会理论性太强，跟现实脱节吗？”

“优秀的学术论文可不会这样，听完我的介绍你就明白了。”

根据笔者的感觉，学术论文的价值主要体现在以下两个方面。

#### ● 提供解决方案

这是优秀的学术论文最基本的能力。学术论文通过提出在设计算法、采用技术等方面的新思路，可以更好地服务于实际需求。实际上，很多企业处理那些悬而未决的问题，很多时候都是采用了优秀学术论文所提出的解决方案。

曾在本书中提到过的分布式存储 Hadoop 技术是 Apache 软件基金会的一个项目，它之所以能够问世并为大量的企业提供分布式存储方案，与 Google 发表的有关 GFS (Google File System)、MapReduce 和 BitTable 等方面的学术论文是分不开的。正是有了这些论文，才使得 Hadoop 这个实现分布式存储的免费技术得到迅速的发展与应用。

#### ● 创业契机

有些时候，优秀学术论文所传达的新知识和新思想，不仅可以解决一些实际问题，还有可能成为在 IT 江湖创立门派的契机。

当今软件世界数一数二的红色巨人 Oracle 在 20 世纪 70 年代的时候，还只是一家小公司。后来在关系型数据模型思想萌生的时候，一篇 IBM 员工发表的有关关系型数据思想的论文被 Oracle 发现后如获至宝，立刻开始进行 RDBMS 的开发，从此才一步步走到了今天。

### 14.4.2 醍醐灌顶，如坐春风

“师兄，学术论文的含金量还真是高啊，不过我就不明白了，学术论文和其他书籍不都属于技术著作吗？它有什么独特的地方值得我们如此留意呢？”

“当然有啦，如果看书和阅读优秀的学术论文效果相同的话，那我还费尽口舌跟你说道半天做什么啊，学术论文必然是有着其独特魅力的。”

的确，学术论文只是目前软件开发行业技术交流的一种媒介，为什么选择学术论文呢？下面简单列举几个学术论文的特点。

- 直入主题

学术论文的格式很简单，就是单刀直入地把一个问题掰开了揉碎了讲个明白，所以不像技术书籍那样会有些开场白或者过起渡作用的边缘文字，也不会如何在增加论文的趣味性和可读性上下太多的工夫，学术论文就是来讲问题的，所以读起来可能也会比较吃力。

- 去掉糟粕后的精华

一般情况下，学术论文的篇幅不能太长，至少要短于一本书的长度，学术论文的这个特性使其必须要在有限的篇幅里（通常不超过几十页）将所要表达的内容交代清楚。于是自顾不暇之时，自然就不会有其他的一些用处不大或无关紧要的糟粕掺杂在里面了。

如果面对的是一篇优秀的学术论文，读者享受到的绝对是最精炼的思想、最简洁的描述、最直白的分析……浓缩的都是精华，学术论文就是思想的浓缩。

- 时代性

不像出一本技术书籍可能需要进行很长时间的周转，发表学术论文的代价要小很多。这也就意味着学术论文可以是最最新的技术产物，而且往往一种新的技术诞生后，出书是很不理智的。因为几乎没有多少观众会来喝彩，倒是学术论文提供了一个快速简洁的宣传窗口。

所以，能够快速地分享到这些时效性很强的论文，对于开发人员来说，必然是一笔不小的技术财富。就算当下用不到，至少也算是明白这种技术的少数人之一了，岂不是很快活？

- 唯一性

并不是所有在学术论文中出现的東西，也肯定会在其他书中呈现。书中讲述的是如何把某件事做出来，而学术论文中可能会研究如何将其做得更好，如某本书上面讲述了如何将文件压缩，而某篇学术论文可能会介绍如何将压缩比达到最高。


由于短期内不会立刻有相关的技术书籍问世，所以在某个时期，学术论文是阐述这方面理论的黄金手册。而且有些时候，由于一些原因，某项技术只会有学术论文这一种理论形式存在，此时读懂学术论文就显得更有必要了。

“书本上的知识往往更像是铁板上钉钉，而学术论文介绍的思想则更富有活力，二者就像是对于同一种食物的不同吃法，书籍会保证营养不流失，而学术论文在保有营养的基础之上，还提供了许多新奇吃法。”

“是啊，如果把学习书本上的知识比作写信的话，那么学术论文的方便性就像是网络聊天。”

“呵呵，有关学术论文的比喻还真不少啊。”

“学术论文是一种非常棒的技术交流媒介，在 Java 界有很多学术论文都是开发人员必须要知道的，所以你以后要注意培养自己阅读学术论文的习惯啊！”

 **提示** 本书中不止一次地提到过，IT 行业的新发展一般在欧美。因此，笔者强烈建议想尝鲜的读者朋友们去读一些原版的学术论文，例如 ACM 的学术论文，其含金量都极高。

## 14.5 本章小结

作为本书的最后一章，本章与读者探讨了一些 Java 开发人员在职场中需要注意的杂项，其中既有与 Java 技术联系紧密的数理知识和学术论文，也有稍微次要一些的方面如专业英语，同时也将本书一直以来倡导的重视积累的工作习惯做了深入的探讨。到这里本书也就结束了，希望笔者多年的一些经验与感悟会对读者朋友们有所帮助。